Δομές Δεδομένων σε C

Μάθημα 4:

Απλά Συνδεδεμένη Λίστα

Δημήτρης Ψούνης

Περιεχόμενα Μαθήματος

Α. Θεωρία

1. Λίστα
   1. Ορισμός Λίστας
   2. Βασικές Πράξεις
2. Απλά Συνδεδεμένη Λίστα
   1. Γενικά
   2. Υλοποίηση σε C: Δηλώσεις
   3. Υλοποίηση σε C: Αρχικοποίηση
   4. Υλοποίηση σε C: Κενή Λίστα
   5. Υλοποίηση σε C: Περιεχόμενο Κόμβου
   6. Υλοποίηση σε C: Εισαγωγή στην αρχή
   7. Υλοποίηση σε C: Εισαγωγή μετά από κόμβο
   8. Υλοποίηση σε C: Διαγραφή στην αρχή
   9. Υλοποίηση σε C: Διαγραφή μετά από κόμβο
   10. Υλοποίηση σε C: Καταστροφή Λίστας
   11. Υλοποίηση σε C: Εκτύπωση Λίστας
3. Ακολουθιακή Λίστα
   1. Γενικά
   2. Υλοποίηση σε C: Δηλώσεις
   3. Υλοποίηση σε C: Αρχικοποίηση
   4. Υλοποίηση σε C: Κενή Λίστα
   5. Υλοποίηση σε C: Περιεχόμενο Κόμβου
   6. Υλοποίηση σε C: Εισαγωγή
   7. Υλοποίηση σε C: Διαγραφή
   8. Υλοποίηση σε C: Εκτύπωση Λίστας
   9. Παρατηρήσεις

Β.Ασκήσεις

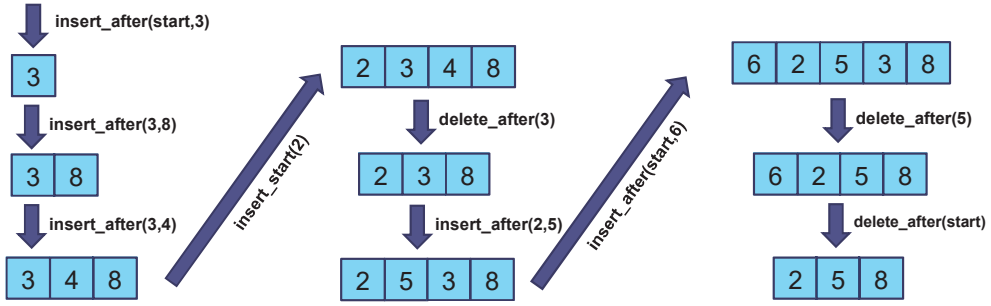
Α. Θεωρία

1. Λίστα

1. Ορισμός Λίστας

Η **«Λίστα»** είναι μια δομή δεδομένων με γραμμική διάταξη στην οποία:

- Η προσθήκη (insert) ενός στοιχείου, γίνεται μετά από κάποιο υφιστάμενο στοιχείο της λίστας
- Η διαγραφή (delete) ενός στοιχείου, γίνεται μετά από κάποιο υφιστάμενο στοιχείο της λίστας



Παρατήρηση:

- Η λίστα είναι χρήσιμη σε οποιαδήποτε δεδομένα ορίζονται από τη σειρά τους!
- Είναι «χρησιμότερη» από τον πίνακα διότι δεν έχει ένα συγκεκριμένο μέγεθος (χωρητικότητα).

Α. Θεωρία

1. Λίστα

2. Βασικές Πράξεις

Οι **βασικές πράξεις** σε μία λίστα είναι:

- **Αρχικοποίηση** της λίστας (**init**)
- **Εισαγωγή** ενός στοιχείου στη λίστα (**insert**)
- **Διαγραφή** ενός στοιχείου από τη λίστα (**delete**)
- **Έλεγχος** αν η λίστα είναι άδεια (**empty**)
- **Περιεχόμενο** ενός κόμβου της λίστας (**data**)

Υπάρχουν δύο υλοποιήσεις:

- Με δείκτες
  - Την ονομάζουμε «απλά συνδεδεμένη λίστα»
- Με πίνακα
  - Την ονομάζουμε «ακολουθιακή λίστα»

Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
1. Εισαγωγή

Στην πρώτη υλοποίηση (**απλά συνδεδεμένη λίστα**) θα χρησιμοποιήσουμε:
- Ο κόμβος θα αποτελείται από τα δεδομένα (data) και τον δείκτη στον επόμενο (next)
- Ένας δείκτης θα είναι η αρχή της λίστας (συνηθίζεται να ονομάζεται head)

Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
2. Υλοποίηση σε C: Δηλώσεις

Οι **δηλώσεις** σε C είναι οι ακόλουθες:
- Ο κόμβος της λίστας είναι μία δομή (struct) με τα εξής στοιχεία:
  - Το data μέρος του κόμβου (σε τύπο δεδομένων που ορίζουμε)
  - Τον δείκτη next που δείχνει το επόμενο στοιχείο της λίστας.

```
typedef int elem;          /* typos dedomenwn listas */

struct node{               /* Typos komvou listas */
   elem data;              /* dedomena */
   struct node *next;      /* epomenos */
};

typedef struct node LIST_NODE; /* Sinwnimo tou komvou listas */
typedef struct node *LIST_PTR; /* Sinwnimo tou deikti komvou */
```

Η λίστα θα είναι ένας δείκτης σε κόμβο λίστας (θα δηλώνεται στη main).

Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
3. Υλοποίηση σε C: Αρχικοποίηση Λίστας

Η αρχικοποίηση γίνεται θέτοντας τον δείκτη λίστας ίσο με NULL

```
/* LL_init(): arxikopoiei tin lista */
void LL_init(LIST_PTR *head)
{
   *head=NULL;
}
```

Προσοχή:
- Πάντα προτού ξεκινάμε την χρήση της λίστας θα πρέπει να καλούμε μία φορά αυτήν τη συνάρτηση!

Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
4. Υλοποίηση σε C: Κενή Λίστα

Ο **έλεγχος** αν η λίστα είναι **κενή**, γίνεται βλέποντας αν ο δείκτης αρχής λίστας είναι ίσος με NULL.

```
/* LL_empty(): epistrefei TRUE/FALSE
 *          analoga me to an i lista einai adeia */
int LL_empty(LIST_PTR head)
{
   return head == NULL;
}
```

Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
5. Υλοποίηση σε C: Περιεχόμενο Κόμβου

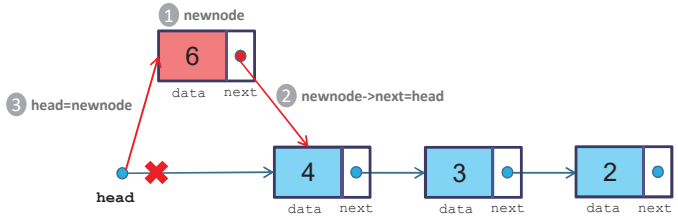
Η συνάρτηση επιστρέφει το περιεχόμενο (τα δεδομένα) ενός κόμβου.

```
/* LL_data(): epistrefei ta dedomena tou komvou
              pou deixnei o deiktis p */
elem LL_data(LIST_PTR p)
{
   return p->data;
}
```


Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
6. Υλοποίηση σε C: Εισαγωγή στην Αρχή

Η συνάρτηση εισάγει έναν νέο κόμβο στην αρχή της λίστας:
1. Δημιουργεί τον νέο κόμβο και θέτει τα δεδομένα σε αυτόν
2. Θέτει τον επόμενο του κόμβου να δείχνει εκεί που δείχνει η κεφαλή της λίστας
3. Θέτει την κεφαλή της λίστας να δείχνει στον νέο κόμβο

ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΡΧΗ του στοιχείου «6» :



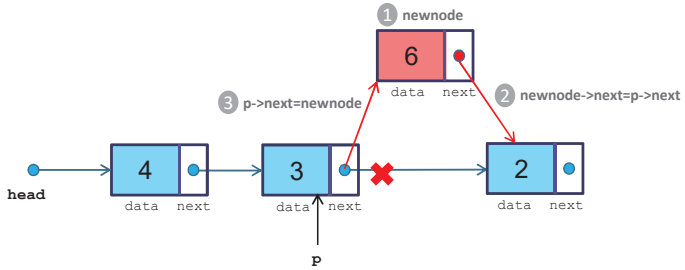
Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
6. Υλοποίηση σε C: Εισαγωγή στην Αρχή

```
/* LL_insert_start(): Eisagei to stoixeio x
                      stin arxi tis listas */
int LL_insert_start(LIST_PTR *head,elem x)
{
   LIST_PTR newnode;

   newnode=(LIST_NODE *)malloc(sizeof(LIST_NODE));
   if (!newnode)
   {
      printf("Adynamia desmeusis mnimis");
      return FALSE;
   }
   newnode->data=x;

   newnode->next=*head;
   *head=newnode;
   return TRUE;
}
```


Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
7. Υλοποίηση σε C: Εισαγωγή Μετά από Κόμβο

Η συνάρτηση εισάγει έναν νέο κόμβο μετά από έναν ενδιάμεσο κόμβο (στον οποίο δείχνει ο p):
1. Δημιουργεί τον νέο κόμβο και θέτει τα δεδομένα σε αυτόν.
2. Θέτει τον next του νέου κόμβου να δείχνει στο next του ενδιάμεσου κόμβου.
3. Θέτει το next του ενδιάμεσου κόμβου να δείχνει στο νέο κόμβο.

ΕΙΣΑΓΩΓΗ ΜΕΤΑ από τον κόμβο με στοιχείο «3», του στοιχείου «6» :



Παρατήρηση:
- Ο αλγόριθμος είναι σωστός ακόμη κι αν ο p δείχνει στο τελευταίο στοιχείο της λίστας

Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
7. Υλοποίηση σε C: Εισαγωγή Μετά από Κόμβο

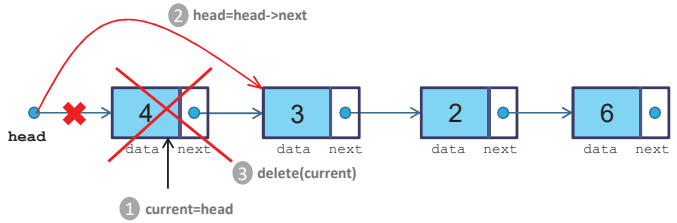
```
/* LL_insert_after(): Eisagei to stoixeio x
                      meta to stoixeio pou deixnei o p */
int LL_insert_after(LIST_PTR p,elem x)
{
        LIST_PTR newnode;

        newnode=malloc(sizeof(LIST_NODE));
        if (!newnode)
        {
                printf("Adynamia desmeusis mnimis");
                return FALSE;
        }
        newnode->data=x;

        newnode->next=p->next;
        p->next=newnode;
        return TRUE;
}
```


Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
8. Υλοποίηση σε C: Διαγραφή στην αρχή

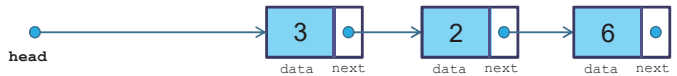
Η συνάρτηση δέχεται ως όρισμα την κεφαλή μιας λίστας και διαγράφει τον πρώτο κόμβο της λίστας:
1. Θέτει έναν δείκτη current να δείχνει στον πρώτο κόμβο
2. Θέτει την κεφαλή της λίστας να δείχνει στον επόμενο κόμβο
3. Διαγράφει τον κόμβο που δείχνει ο current

ΔΙΑΓΡΑΦΗ του κόμβου ΜΕΤΑ τον prev

ΠΡΙΝ:



ΜΕΤΑ:



Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
8. Υλοποίηση σε C: Διαγραφή στην αρχή

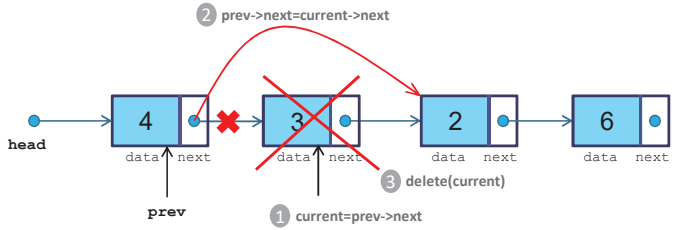
```
/* LL_delete_start(): Diagrafei ton komvo poy deixnei
        i kefali tis listas */
int LL_delete_start(LIST_PTR *head, elem *x)
{
   LIST_PTR current;

   if (*head==NULL)
      return FALSE;

   current=*head;
   *x=current->data;

   (*head)=(*head)->next;
   free(current);
   return TRUE;
}
```


Α. Θεωρία
2. Απλά Συνδεδεμένη Λίστα
9. Υλοποίηση σε C: Διαγραφή μετά από κόμβο

Η συνάρτηση δέχεται ως όρισμα έναν κόμβο (στον οποίο δείχνει ο prev) και διαγράφει τον επόμενο κόμβο του:
1. Θέτει έναν δείκτη current να δείχνει στον επόμενο του prev
2. Θέτει τον επόμενο του prev να δείχνει στον επόμενο του current
3. Διαγράφει τον κόμβο που δείχνει ο current

ΔΙΑΓΡΑΦΗ του κόμβου ΜΕΤΑ τον prev

ΠΡΙΝ:



ΜΕΤΑ:

Α. Θεωρία

2. Απλά Συνδεδεμένη Λίστα

9. Υλοποίηση σε C: Διαγραφή μετά από κόμβο

```
/* LL_delete_after(): Diagrafei ton epomeno tou
                                komvou poy deixnei o prev */
int LL_delete_after(LIST_PTR prev, elem *x)
{
   LIST_PTR current;

   if (prev->next==NULL)
      return FALSE;

   current=prev->next;
   *x=current->data;

   prev->next=current->next;
   free(current);
   return TRUE;
}
```


Α. Θεωρία

2. Απλά Συνδεδεμένη Λίστα

10. Υλοποίηση σε C: Καταστροφή Λίστας

Η συνάρτηση δέχεται ως όρισμα την κεφαλή μιας λίστας και διαγράφει όλους τους κόμβους της (χρήσιμη για την αποδέσμευση της μνήμης στο τέλος του προγράμματος)

```
/* LL_destroy(): Apodesmeyei to xwro poy exei desmeusei i lista       */

void LL_destroy(LIST_PTR *head)
{
   LIST_PTR ptr;

   while (*head!=NULL)
   {
      ptr=*head;
      *head=(*head)->next;
      free(ptr);
   }
}
```

Παρατήρηση:
- Αποτελεί προγραμματιστική υποχρέωση να αποδεσμευτεί ο χώρος που έχει δεσμεύσει η λίστα.

Α. Θεωρία

2. Απλά Συνδεδεμένη Λίστα

11. Υλοποίηση σε C: Εκτύπωση Λίστας

Η συνάρτηση δέχεται ως όρισμα την κεφαλή μιας λίστας και τυπώνει το περιεχόμενο των κόμβων της (εδώ θεωρούμε ότι είναι λίστα ακεραίων)

```
/* LL_print(): Typwnei ta periexomena mias syndedemenis listas        */

void LL_print(LIST_PTR head)
{
   LIST_PTR current;

   current=head;
   while(current!=NULL)
   {
      printf("%d ",current->data);
      current=current->next;
   }
}
```

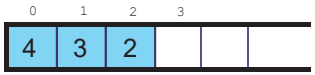
Παρατήρηση:
- Η διαπέραση μιας λίστας ώστε να γίνεται μια ενέργεια, είναι πολύ συχνή στις συνδεδεμένες λίστες. Οι εφαρμογές στις ασκήσεις θα αναδείξουν αυτήν τη χρήση!

Α. Θεωρία

3. Ακολουθιακή Λίστα

1. Εισαγωγή

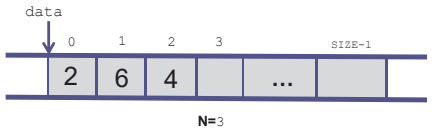
Στην δεύτερη υλοποίηση (**ακολουθιακή λίστα**) θα χρησιμοποιήσουμε:
- Έναν πίνακα που θα αποθηκεύει τα στοιχεία
  - Η διάταξη των στοιχείων θα είναι σύμφωνα με την αρίθμηση του πίνακα
- Σε κάθε θέση του πίνακα θα αποθηκεύεται ένα στοιχείο



Α. Θεωρία
3. Ακολουθιακή Λίστα
2. Υλοποίηση σε C: Δηλώσεις

Οι **δηλώσεις** σε C είναι οι ακόλουθες:
- Η λίστα είναι μία δομή (struct) με τα εξής στοιχεία:
  - Τον πίνακα των δεδομένων (data)
  - Το πλήθος των στοιχείων (N)

```
typedef int elem;          /* typos dedomenwn listas */

typedef struct list LIST;  /* Sinwnimo tis listas */

#define SIZE 100
struct list{               /* Akolouthiaki list */
   elem data[SIZE];        /* Pinakas Dedomenwn */
   int N;                  /* plithos stoixeiwn */
};
```



Α. Θεωρία
3. Ακολουθιακή Λίστα
3. Υλοποίηση σε C: Αρχικοποίηση Λίστας

Η αρχικοποίηση γίνεται θέτοντας το πλήθος των στοιχείων ίσο με 0

```
/* SL_init(): arxikopoiei tin lista */
void SL_init(LIST *l)
{
   l->N=0;
}
```

Προσοχή:
- Πάντα προτού ξεκινάμε την χρήση της λίστας θα πρέπει να καλούμε μία φορά αυτήν τη συνάρτηση!

Α. Θεωρία
3. Ακολουθιακή Λίστα
4. Υλοποίηση σε C: Κενή Λίστα

Ο **έλεγχος** αν η λίστα είναι **κενή**, γίνεται βλέποντας αν το πλήθος των στοιχείων είναι 0.

```
/* SL_empty(): epistrefei TRUE/FALSE
 *          analoga me to an i lista einai adeia */
int SL_empty(LIST l)
{
        return l.N == 0;
}
```


Α. Θεωρία
3. Ακολουθιακή Λίστα
5. Υλοποίηση σε C: Περιεχόμενο Κόμβου

Η συνάρτηση επιστρέφει το περιεχόμενο (τα δεδομένα) ενός κόμβου.

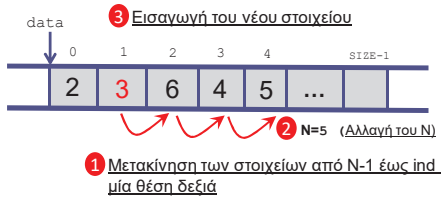
```
/* SL_data(): epistrefei ta dedomena tou komvou
                            tou index ind */
elem SL_data(LIST l, int ind)
{
        return l.data[ind];
}
```

Α. Θεωρία
3. Ακολουθιακή Λίστα
6. Υλοποίηση σε C: Εισαγωγή

Η συνάρτηση εισάγει έναν νέο στοιχείο elem στη θέση ind στη λίστα:
1. Μετακινεί όλα τα στοιχεία από τη θέση ind, μία θέση δεξιά
2. Αλλάζει το πλήθος στοιχείων
3. Εισάγει το νέο στοιχείο στη θέση ind

ΕΙΣΑΓΩΓΗ του στοιχείου «3» στη θέση 1:

ΜΕΤΑ:



Α. Θεωρία
3. Ακολουθιακή Λίστα
6. Υλοποίηση σε C: Εισαγωγή

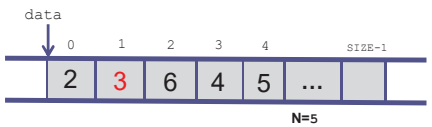
```
/* SL_insert(): Eisagei to stoixeio x sti thesi ind */
int SL_insert(LIST *l, int ind, elem x)
{
   int i;

   if (ind<0 || ind>l->N)
      return FALSE;

   if (l->N < SIZE)
   {
      for (i=l->N; i>ind; i--)
         l->data[i]=l->data[i-1];

      l->data[ind]=x;
      l->N++;
      return TRUE;
   }
   else
      return FALSE;
}
```


Α. Θεωρία
3. Ακολουθιακή Λίστα
7. Υλοποίηση σε C: Διαγραφή

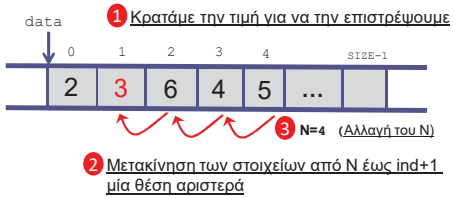
Η συνάρτηση δέχεται ως όρισμα τη θέση ind και διαγράφει το στοιχειό που βρίσκεται σε αυτήν
1. Κρατάμε το στοιχείο στη θέση για να το επιστρέψουμε.
2. Μετακίνηση των στοιχείων από N έως ind+1 μία θέση αριστερά.
3. Μείωση του N κάτα 1.

ΔΙΑΓΡΑΦΗ του στοιχείου στη θέση 1

ΠΡΙΝ:



ΜΕΤΑ:



Α. Θεωρία
3. Ακολουθιακή Λίστα
7. Υλοποίηση σε C: Διαγραφή

```
/* SL_delete(): Diagrafei ta data pou vriskontai
   sti thesi ind */
int SL_delete(LIST *l, int ind, elem *x)
{
   int i;

   if (ind<0 || ind>=l->N)
      return FALSE;

   *x = l->data[ind];

   for (i=ind; i<l->N; i++)
      l->data[i]=l->data[i+1];

   l->N--;
   return TRUE;

}
```

Α. Θεωρία
3. Ακολουθιακή Λίστα
8. Παρατηρήσεις

- Η ακολουθιακή λίστα δεν είναι ιδιαίτερα δημοφιλής:
  - Μία εισαγωγή ή διαγραφή είναι χρονοβόρα, διότι μπορεί να απαιτεί ακόμη και την μετακίνηση όλων των στοιχείων του πίνακα.
  - Επίσης έχει σταθερό μέγιστο μέγεθος του πίνακα, οπότε υπάρχει ένας περιορισμός σε σχέση με τη συνδεδεμένη λίστα.

- Έτσι ως υλοποίηση της λίστας προτιμάται η απλά συνδεδεμένη λίστα
  - Παρά το ότι απαιτεί παραπάνω χρόνο για την εύρεση ενός στοιχείου (ενώ στην ακολουθιακή λίστα, είναι πολύ γρήγορη)

Β. Ασκήσεις
Εφαρμογή 1.1: Λίστα Εγγραφών

- Κάνετε κατάλληλη τροποποίηση στην απλά συνδεδεμένη λίστα, έτσι ώστε να αποτελεί μία λίστα εγγραφών φοιτητών.
  - Κάθε εγγραφή θα αποτελείται από μία συμβολοσειρά 80 χαρακτήρων (name) και έναν ακέραιο αριθμό (degree).
  - Τροποποιήστε τις βασικές πράξεις και ιδιαίτερα την πράξη της εκτύπωσης ώστε να παράγει μία μορφοποιημένη εκτύπωση των στοιχείων της λίστας.

Β. Ασκήσεις
Εφαρμογή 1.2: Συνάρτηση Εισαγωγής

- Κατασκευάστε την (δευτερεύουσα) πράξη της εισαγωγής (LL_insert) έτσι ώστε η λίστα να διατηρείται ταξινομημένη σε αύξουσα αλφαβητική σειρά του ονόματος.
  - Πραγματοποιήστε έλεγχο ορθότητας με κατάλληλες εκτυπώσεις στην main.

Β. Ασκήσεις
Εφαρμογή 1.3: Συνάρτηση Διαγραφής

- Κατασκευάστε την (δευτερεύουσα) πράξη της διαγραφής (LL_delete) η οποία να δέχεται ως όρισμα τη λίστα και το όνομα ενός φοιτητή.
- Να αναζητεί την εγγραφή και να διαγράφει το φοιτητή (αν αυτός υπάρχει)

Β. Ασκήσεις
Εφαρμογή 1.4: Μέσος Όρος Βαθμολογίας

- Κατασκευάστε την συνάρτηση LL_average η οποία δέχεται ως όρισμα μία λίστα εγγραφών φοιτητών και επιστρέφει τον μέσο όρο των βαθμών των μαθητών.

Β. Ασκήσεις
Εφαρμογή 1.5: Πλήθος Επιτυχόντων

- Κατασκευάστε την συνάρτηση LL_pass η οποία δέχεται ως όρισμα μία λίστα εγγραφών φοιτητών και επιστρέφει το πλήθος των φοιτητών που πέρασαν το μάθημα (Βαθμός >=5).

Β. Ασκήσεις
Εφαρμογή 1.6: Μενού Επιλογών

- Κατασκευάστε main που να υλοποιεί μενού επιλογών των ενεργειών που κατασκευάσαμε:
  - Εισαγωγή φοιτητή
  - Διαγραφή φοιτητή
  - Εκτύπωση Λίστας
  - Μέσος Όρος Βαθμολογίας
  - Πλήθος Επιτυχόντων.
  - Έξοδος

Β. Ασκήσεις
Εφαρμογή 2: Υλοποίηση στοίβας με συνδεδεμένη λίστα

- Υλοποιήστε την στοίβα, χρησιμοποιώντας μία απλά συνδεδεμένη λίστα για την αποθήκευση των δεδομένων