Δομές Δεδομένων σε C

Μάθημα 5:

Παραλλαγές της Συνδεδεμένης Λίστας

Δημήτρης Ψούνης

www.psounis.gr

Περιεχόμενα Μαθήματος

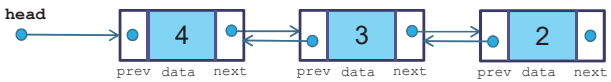
Α. Θεωρία

1. Διπλά Συνδεδεμένη Λίστα

1. Εισαγωγή

Η **διπλά συνδεδεμένη λίστα** είναι μία συνδεδεμένη λίστα που κάθε κόμβος έχει δείκτη και προς τον προηγούμενο του:

- Ο κόμβος θα αποτελείται από τα δεδομένα (data), τον δείκτη στον επόμενο (next) και τον δείκτη στον προηγούμενο (prev)
- Ένας δείκτης θα είναι η αρχή της λίστας (συνηθίζεται να ονομάζεται head)



Α. Θεωρία

1. Διπλά 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.
  - Τον δείκτη prev που δείχνει το προηγούμενο στοιχείο της λίστας.

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

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

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

1. Διπλά Συνδεδεμένη Λίστα

3. Υλοποίηση σε C: Αρχικοποίηση Λίστας

Η αρχικοποίηση γίνεται θέτοντας τον δείκτη λίστας ίσο με NULL

```
/* DLL_init(): arxikopoiei tin lista */
void DLL_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

1. Διπλά 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.

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


Α. Θεωρία

1. Διπλά Συνδεδεμένη Λίστα

5. Υλοποίηση σε C: Περιεχόμενο Κόμβου

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

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

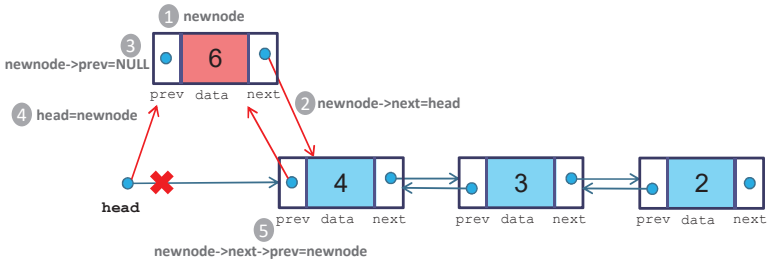

Α. Θεωρία

1. Διπλά 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 τον προηγούμενο του κόμβου να είναι NULL
4. Θέτει την κεφαλή της λίστας να δείχνει στον νέο κόμβο
5. Αν υπάρχει επόμενος κόμβος, τότε ο προηγούμενός του πρέπει να είναι ο νέος κόμβος

ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΡΧΗ του στοιχείου «6» :



Α. Θεωρία

1. Διπλά Συνδεδεμένη Λίστα

6. Υλοποίηση σε C: Εισαγωγή στην Αρχή

```
/* DLL_insert_start(): Eisagei to stoixeio x
                       stin arxi tis listas */
int DLL_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;
    newnode->prev=NULL;
    *head=newnode;
    if newnode->next!=NULL)
       newnode->next->prev=newnode;
    return TRUE;
}
```


Α. Θεωρία

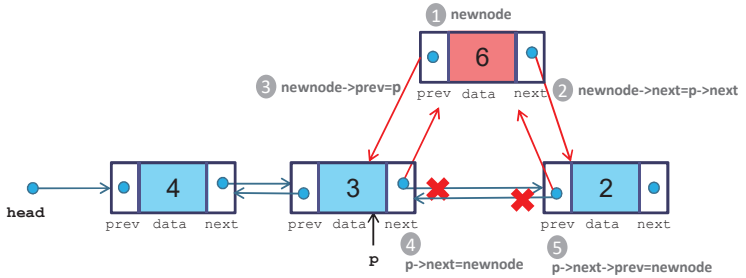
1. Διπλά Συνδεδεμένη Λίστα

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 τον prev του νέου κόμβου να δείχνει στον ενδιάμεσο κόμβο
4. Θέτει το next του ενδιάμεσου κόμβου να δείχνει στο νέο κόμβο.
5. Θέτει το prev του επόμενου κόμβου (αν υπάρχει) 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

1. Διπλά 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 DLL_insert_after(LIST_PTR p,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=p->next;
   newnode->prev=p;
   p->next=newnode;
   if (p->next!=NULL)
      p->next->prev=newnode;
   return TRUE;
}
```


Α. Θεωρία

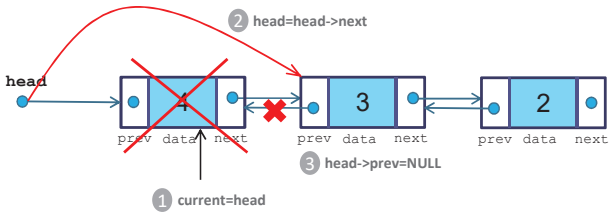
1. Διπλά Συνδεδεμένη Λίστα

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. Θέτει το prev του επόμενου κόμβου (αν υπάρχει) να είναι NULL
4. Διαγράφει τον κόμβο που δείχνει ο current

ΔΙΑΓΡΑΦΗ του πρώτου κόμβου



Α. Θεωρία

1. Διπλά Συνδεδεμένη Λίστα

8. Υλοποίηση σε C: Διαγραφή στην αρχή

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

   if (*head==NULL)
      return FALSE;

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

   (*head)=(*head)->next;
   if ((*head)!=NULL)
      (*head)->prev=NULL;

   free(current);
   return TRUE;
}
```


Α. Θεωρία

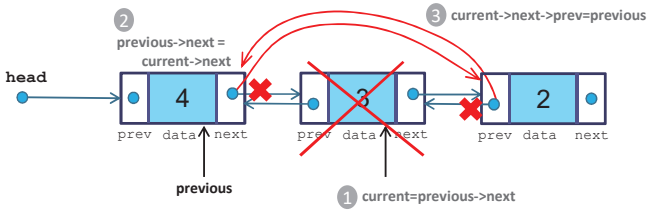
1. Διπλά Συνδεδεμένη Λίστα

9. Υλοποίηση σε C: Διαγραφή μετά από κόμβο

Η συνάρτηση δέχεται ως όρισμα έναν κόμβο (στον οποίο δείχνει ο previous) και διαγράφει τον επόμενο κόμβο του:

1. Θέτει έναν δείκτη current να δείχνει στον επόμενο του previous
2. Θέτει το next του previous να δείχνει στον επόμενο του current
3. Θέτει το prev του επόμενου του current (αν υπάρχει) να δείχνει στον previous
4. Διαγράφει τον κόμβο που δείχνει ο current

ΔΙΑΓΡΑΦΗ του κόμβου ΜΕΤΑ τον prev



Α. Θεωρία

1. Διπλά Συνδεδεμένη Λίστα

9. Υλοποίηση σε C: Διαγραφή μετά από κόμβο

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

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

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

   previous->next=current->next;
   if (current->next!=NULL)
      current->next->prev=previous;

   free(current);
   return TRUE;
}
```


Α. Θεωρία

1. Διπλά 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)

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

void DLL_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
1. Διπλά Συνδεδεμένη Λίστα
11. Υλοποίηση σε C: Εκτύπωση Λίστας

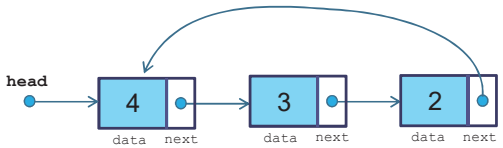
Η συνάρτηση δέχεται ως όρισμα την κεφαλή μιας λίστας και τυπώνει το περιεχόμενο των κόμβων της (εδώ θεωρούμε ότι είναι λίστα ακεραίων)

```
/* DLL_print(): Typwnei ta periexomena mias syndedemenis listas       */
void DLL_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
2. Κυκλική Λίστα
1. Εισαγωγή

Η **κυκλικά συνδεδεμένη λίστα** είναι μία απλά συνδεδεμένη λίστα που ο τελευταίος κόμβος δείχνει στον πρώτο κόμβο:
- 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. Κυκλική 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. Κυκλική Λίστα
3. Υλοποίηση σε C: Αρχικοποίηση Λίστας

Η αρχικοποίηση γίνεται θέτοντας τον δείκτη λίστας ίσο με NULL

```
/* CL_init(): arxikopoiei tin lista */
void CL_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. Κυκλική 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.

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


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

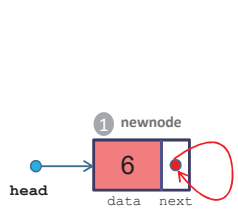
Η συνάρτηση επιστρέφει το περιεχόμενο (τα δεδομένα) ενός κόμβου.

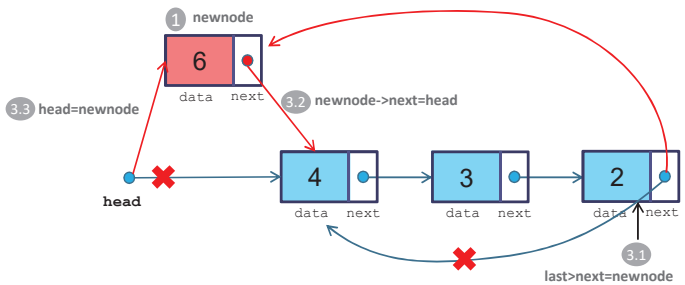
```
/* CL_data(): epistrefei ta dedomena tou komvou
              pou deixnei o deiktis p */
elem CL_data(LIST_PTR p)
{
   return p->data;
}
```


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

Η συνάρτηση εισάγει έναν νέο κόμβο στην αρχή της λίστας:
1. Κατασκευάζει το νέο κόμβο και θέτει τα δεδομένα στο νέο κόμβο.
2. Αν η λίστα είναι άδεια, τότε προσθέτει το νέο κόμβο με αυτόν να δείχνει στον εαυτό του.
3. Αν η λίστα δεν είναι άδεια, τότε:
   1. Βρίσκει τον τελευταίο κόμβο. Το next του τίθεται να δείχνει στο newnode
   2. Το next του νέου κόμβου γίνεται ίσο με το head
   3. Το head δείχνει στο νέο κόμβο.

ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΡΧΗ του στοιχείου «6» :



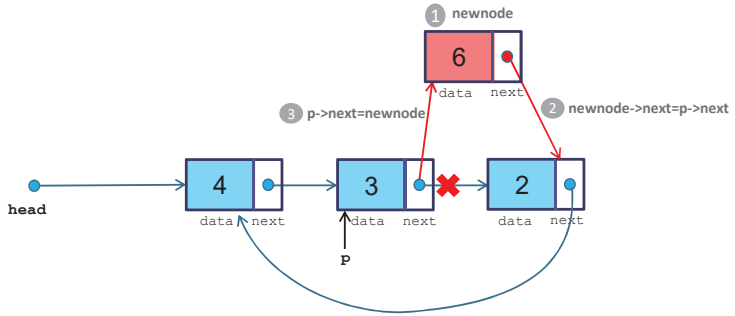


Α. Θεωρία
2. Κυκλική Λίστα
6. Υλοποίηση σε C: Εισαγωγή στην Αρχή

```
/* CL_insert_start(): Eisagei to stoixeio x
stin arxi tis listas */
int CL_insert_start(LIST_PTR *head,elem x)
{
   LIST_PTR newnode, last;
   // 1
   newnode=(LIST_NODE *)
            malloc(sizeof(LIST_NODE));
   if (!newnode)
   {
      printf("Adynamia desmeusis mnimis");
      return FALSE;
   }
   newnode->data=x;

   if (*head == NULL)
   {   // 2
      newnode->next=newnode;
      *head=newnode;
   }
   else
   {
      // 3.1
      last = *head;
      while (last->next!=*head)
          last=last->next;
      last->next=newnode;

      // 3.2
      newnode->next=*head;

      // 3.3
      *head=newnode;
   }

   return TRUE;
}
```

A. Θεωρία
2. Κυκλική 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.



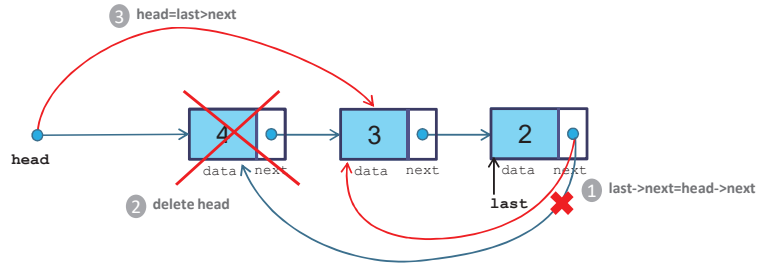
A. Θεωρία
2. Κυκλική Λίστα
7. Υλοποίηση σε C: Εισαγωγή Μετά από Κόμβο

```
/* CL_insert_after(): Eisagei to stoixeio x
                      meta to stoixeio pou deixnei o p */
int CL_insert_after(LIST_PTR p,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=p->next;
   p->next=newnode;
   return TRUE;
}
```


A. Θεωρία
2. Κυκλική Λίστα
8. Υλοποίηση σε C: Διαγραφή πρώτου κόμβου

Η συνάρτηση δέχεται ως όρισμα την κεφαλή μιας λίστας και διαγράφει τον πρώτο κόμβο της λίστας:
1. Αν η λίστα έχει έναν κόμβο, τότε θέτουμε το head ίσο με NULL
2. Αν η λίστα έχει τουλάχιστον δύο κόμβους:
   1. Εύρεση του τελευταίου κόμβου. Θέτουμε το next ίσο με τον επόμενο του head
   2. Διαγράφει τον κόμβο που δείχνει ο head
   3. Θέτουμε το head ίσο με τον επόμενο του last

ΔΙΑΓΡΑΦΗ του κόμβου “4”



A. Θεωρία
2. Κυκλική Λίστα
8. Υλοποίηση σε C: Διαγραφή στην αρχή

```
/* CL_delete_start(): Diagrafei ton komvo
poy deixnei i kefali tis listas */

int CL_delete_start(LIST_PTR *head, elem *x)
{
   LIST_PTR next, last;

   // 0 stoixeia
   if (*head==NULL)
      return FALSE;

   // 1 stoixeio
   if (*head==(*head)->next)
   {
      *x=(*head)->data;
      free(*head);
      *head=NULL;
      return TRUE;
   }
   // >=2 stoixeia
   last=(*head)->next;
   while (last->next!=(*head))
      last=last->next;
   last->next=(*head)->next;

   *x=(*head)->data;
   free(*head);
   (*head)=last->next;
   return TRUE;
}
```

Α. Θεωρία

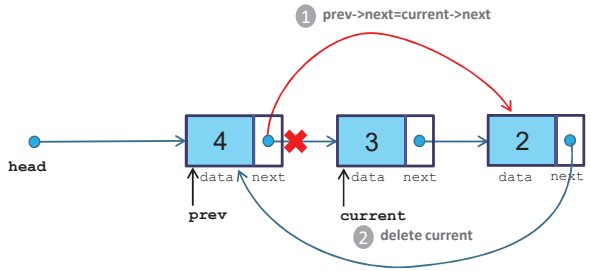
2. Κυκλική Λίστα

9. Υλοποίηση σε C: Διαγραφή επόμενου κόμβου

Η συνάρτηση δέχεται ως όρισμα έναν κόμβο (στον οποίο δείχνει ο current και δεν είναι ο 1ος κόμβος της λίστας) και τον διαγράφει:

1. Βρίσκει τον προηγούμενο κόμβο prev. Θέτει τον επόμενο του prev να δείχνει στον επόμενο του current
2. Διαγράφει τον κόμβο που δείχνει ο current

ΔΙΑΓΡΑΦΗ του κόμβου ΜΕΤΑ τον prev



Α. Θεωρία

2. Κυκλική Λίστα

9. Υλοποίηση σε C: Διαγραφή μετά από κόμβο

```
/* CL_delete_after(): Diagrafei ton komvo
                      poy deixnei o current */
int CL_delete_after(LIST_PTR current, elem *x)
{
   LIST_PTR prev;

   while (prev->next!=current)
      prev=prev->next;

   *x=current->data;

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


Α. Θεωρία

2. Κυκλική 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)

```
/* CL_destroy(): Apodesmeyei to xwro poy exei
                 desmeusei i lista */
void CL_destroy(LIST_PTR *head)
{
   LIST_PTR ptr;

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

Παρατήρηση:

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

Α. Θεωρία

2. Κυκλική 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)

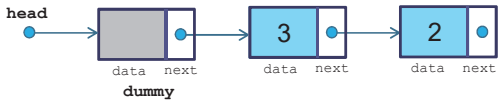
```
/* CL_print(): Typwnei ta periexomena tis listas     */
void CL_print(LIST_PTR head)
{
   LIST_PTR current;

   if (head!=NULL)
   {
       printf("%d ", head->data);

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

Α. Θεωρία

3. Άλλες Μορφές Λίστας

1. Παρατηρήσεις

- Υπάρχουν και άλλες μορφές λίστας:
- Η λίστα με κεφαλή (όπου προστίθεται ένας καινούριος dummy κόμβος στην αρχή για να απλοποιηθούν οι πράξεις της εισαγωγής και διαγραφής)



- Καθώς και συνδυασμοί αυτών:
  - π.χ. κυκλικά διπλά συνδεδεμένη λίστα
- ή άλλες παραλλαγές που μας είναι χρήσιμες κατά περίπτωση
  - π.χ. μια λίστα που έχει δύο δείκτες (έναν που να δείχνει στην αρχή και έναν στο τέλος για να κάνουμε γρήγορες προσθήκες στην αρχή και στο τέλος της λίστας)

Β. Ασκήσεις

Εφαρμογή 1: Αντίστροφη εκτύπωση

- Επεκτείνετε την διπλά συνδεδεμένη λίστα με την πράξη print_reverse η οποία παίρνει σαν όρισμα μία λίστα και τυπώνει τα περιεχόμενά της με αντίστροφη σειρά

Β. Ασκήσεις

Εφαρμογή 2: i-οστό στοιχείο λίστας

- Επεκτείνετε την κυκλικά συνδεδεμένη λίστα με την πράξη get_i η οποία παίρνει σαν όρισμα μία λίστα και επιστρέφει το i-οστό στοιχείο της.
- Θεωρήστε ότι η αρίθμηση των στοιχείων της λίστας ξεκινά από το 1
- Προσοχή, ο δείκτης ενσωματώνει την κυκλικότητα. Π.χ. αν η λίστα έχει 10 στοιχεία και ζητήσει το 26ο στοιχείο, τότε να επιστρέφεται το 6ο στοιχείο.

Β. Ασκήσεις

Εφαρμογή 3: Υλοποίηση ουράς με συνδεδεμένη λίστα

- Υλοποιήστε την ουρά, δημιουργώντας μια κατάλληλη δομή δεδομένων μορφής συνδεδεμένης λίστας ώστε οι πράξεις της εξαγωγής και της εισαγωγής να υλοποιούνται χωρίς να απαιτείται η διαπέραση της λίστας.
  - Hint: Μπορείτε να τροποποιήσετε την δήλωση της λίστας ώστε να μην αποτελείται μόνο από έναν δείκτη που να δείχνει στην αρχή της λίστας.
  - Hint 2: Σκεφθείτε ποιες πράξεις πρέπει να γίνονται στην ουρά και έπειτα παρατηρείστε ποια από τις υλοποιήσεις της λίστας που είδαμε στη θεωρία είναι η γρηγορότερη γι΄ αυτές τις πράξεις.

Β. Ασκήσεις

Εφαρμογή 4: Μία λίστα από ουρές

- Κατασκευάστε μία δομή δεδομένων που θα είναι χρήσιμη για την προσομοίωση των ταμείων π.χ. ενός σούπερμάρκετ.
- Συγκεκριμένα γράψτε τις δηλώσεις που απαιτούνται για να κατασκευάσετε μια διπλά συνδεδεμένη λίστα, της οποίας κάθε στοιχείο θα είναι μια ουρά ακεραίων.

Ελέγξτε την ορθότητα της δομής σας με μία κατάλληλη main.