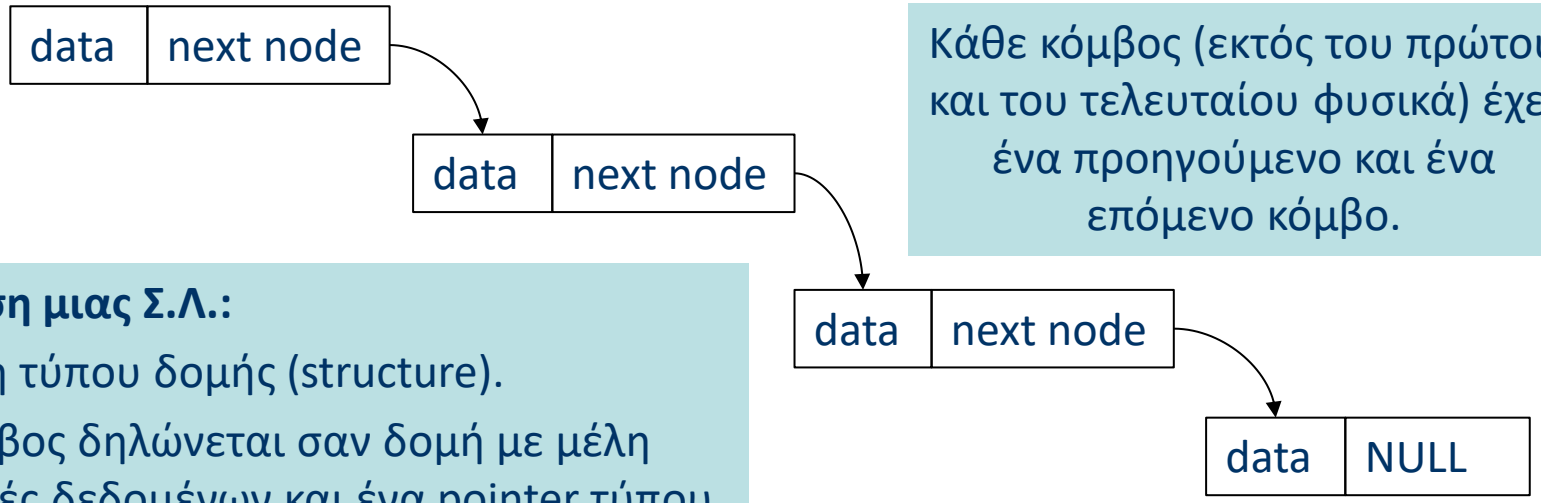
Η γλώσσα προγραμματισμού C

Συνδεδεμένες Λίστες

Τι είναι οι συνδεδεμένες λίστες (linked lists)

- Μια Συνδεδεμένη Λίστα (Σ.Λ.) είναι μια ακολουθία από κόμβους που συνδέονται σειριακά και που μπορούμε να τους διαχειριστούμε δυναμικά.
- Κάθε κόμβος, εκτός των δεδομένων (data) που περιέχει, περιλαμβάνει και ένα τουλάχιστον δείκτη που δείχνει σε διεύθυνση άλλου κόμβου.
- Αν ο δείκτης κάποιου κόμβου είναι μηδενικός (NULL) τότε αυτός είναι ο τελευταίος (ή και ο πρώτος –ανάλογα πως διατρέχεται η λίστα) κόμβος.



Κάθε κόμβος (εκτός του πρώτου και του τελευταίου φυσικά) έχει ένα προηγούμενο και ένα επόμενο κόμβο.

Υλοποίηση μιας Σ.Λ.:

Με χρήση τύπου δομής (structure).

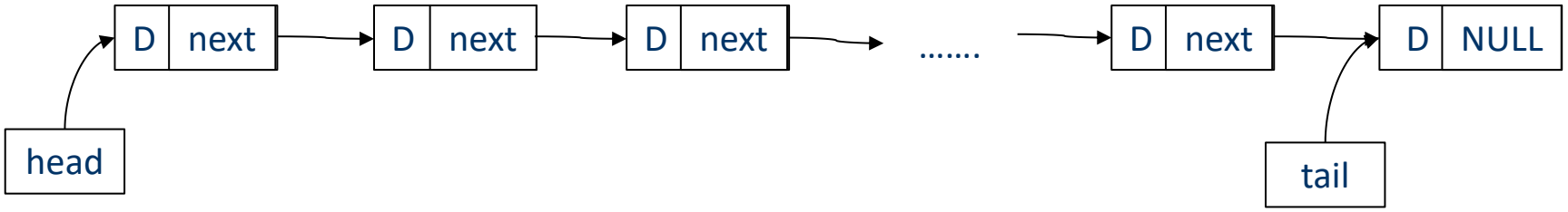
Κάθε κόμβος δηλώνεται σαν δομή με μέλη μεταβλητές δεδομένων και ένα pointer τύπου της δομής (που δείχνει στο επόμενο κόμβο

Πλεονεκτήματα – Μειονεκτήματα των Σ.Λ.

- Πλεονεκτήματα :
  - Μπορούμε να προσθέσουμε ή ν' αφαιρέσουμε κόμβους σε οποιοδήποτε σημείο της διάταξης (ή και ν' αλλάξουμε τη θέση ενός κόμβου).
  - Το πλήθος των κόμβων είναι δυναμικά αυξομειούμενο και δεν χρειάζεται να το προσδιορίζουμε εξ αρχής.
  - Οι κόμβοι μιας Σ.Λ. δεν χρειάζεται να τοποθετηθούν σε διαδοχικές θέσεις μνήμης όπως συμβαίνει με του πίνακες (λογική σειρά ≠ φυσική σειρά).
- Μειονεκτήματα:
  - Δεν έχουμε άμεση πρόσβαση σε κάποιο κόμβο της διάταξης (random access). Θα πρέπει να επισκεφτούμε διαδοχικά τους προηγούμενους κόμβους μέχρι να φτάσουμε στον κόμβο που θέλουμε.
  - Στη C απαιτούνται εντολές για δυναμική διαχείριση μνήμης και δείκτες, που σημαίνει πολύπλοκα προγράμματα και αυξημένες πιθανότητες λάθους.
  - Επίσης, υπάρχει μεγαλύτερο υπολογιστικό κόστος, καθώς οι κόμβοι δεσμεύονται δυναμικά και απαιτείται ένας δείκτης για κάθε κόμβο.

Απλά Συνδεδεμένες Λίστες (Α.Σ.Λ.)

- Κάθε κόμβος έχει ένα δείκτη ο οποίος δείχνει στον επόμενο κόμβο.
- Ο δείκτης του τελευταίου κόμβου δείχνει στο NULL (μηδενικός).



- Για τη ευκολότερη διαχείριση μιας Α.Σ.Λ. χρησιμοποιείται ένας δείκτης που δείχνει στον αρχικό (head) κόμβο.
- Ορισμένες φορές, χρησιμοποιείται και δείκτης και που δείχνει στο τελευταίο (tail) κόμβο.

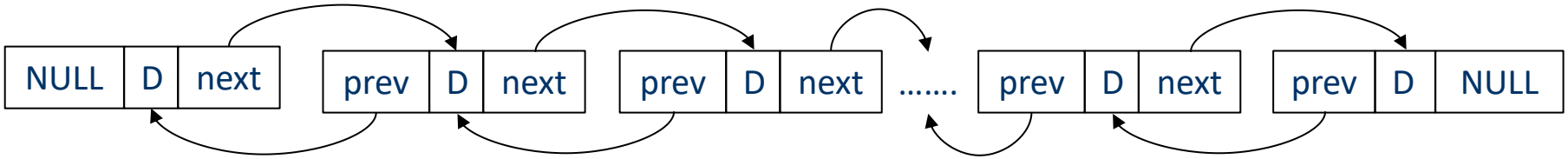
Παράδειγμα δήλωσης κόμβου:

```
typedef struct node {
        int  data;
        struct node * next;
} sll_node;
```

Προφανώς μπορούμε να βάλουμε όσες μεταβλητές θέλουμε για τα δεδομένα.

Διπλά Συνδεδεμένες Λίστες (Δ.Σ.Λ.)

- Έχουμε δύο δείκτες σε κάθε κόμβο:
  - ο ένας δείχνει στον προηγούμενο και
  - ο άλλος στο επόμενο κόμβο.



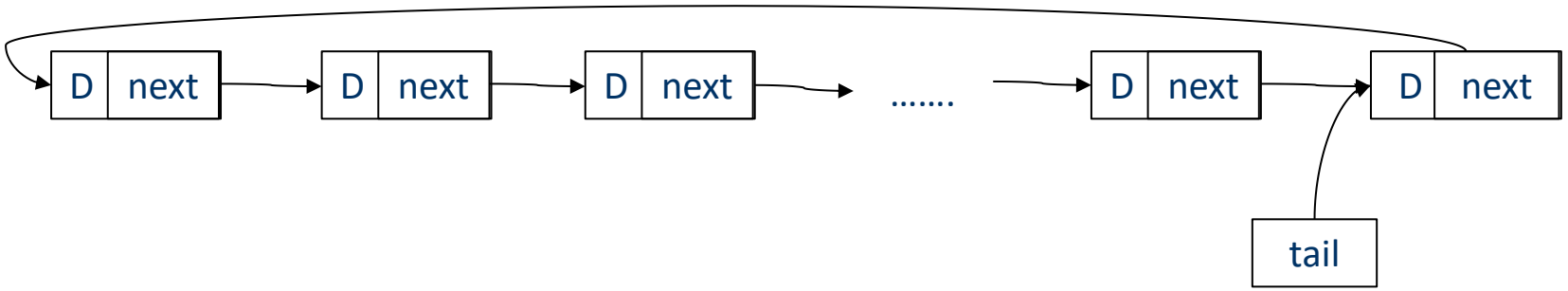
- Δυο επιπλέον πλεονεκτήματα:
  - η Δ.Σ.Λ. μπορεί να διαβαστεί και προς τις δυο κατευθύνσεις και
  - η διάταξη διατρέχεται ακόμα και αν καταστραφεί κάποιος σύνδεσμος.

Παράδειγμα δήλωσης κόμβου:

```
typedef struct node {
        struct node * prev;
         int  data;
         struct node * next;
} dll_node;
```

Κυκλικά Συνδεδεμένες Λίστες (Κ.Σ.Λ.)

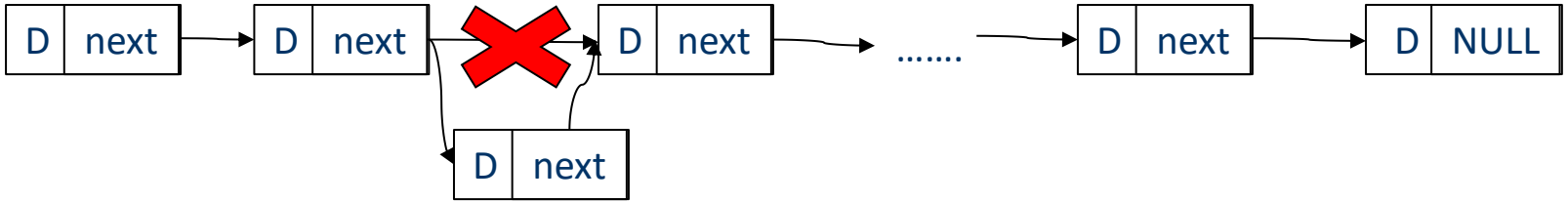
- Ίδιες με τις Α.Σ.Λ με την διαφορά ότι:
  - ο τελευταίος κόμβος δείχνει στον πρώτο αντί στο NULL



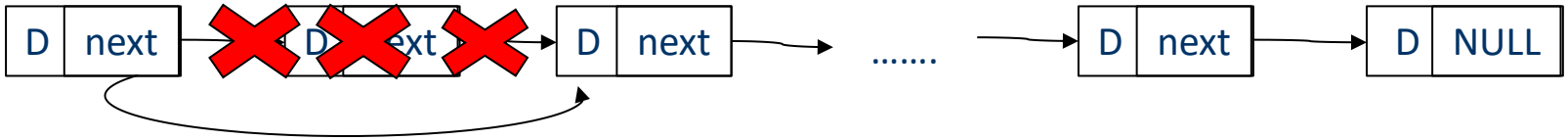
  - Για τη ευκολότερη διαχείριση μιας Κ.Σ.Λ. χρησιμοποιείται ένας δείκτης που δείχνει στον τελευταίο (tail) κόμβο.

Βασικές λειτουργίες σε Σ.Λ.

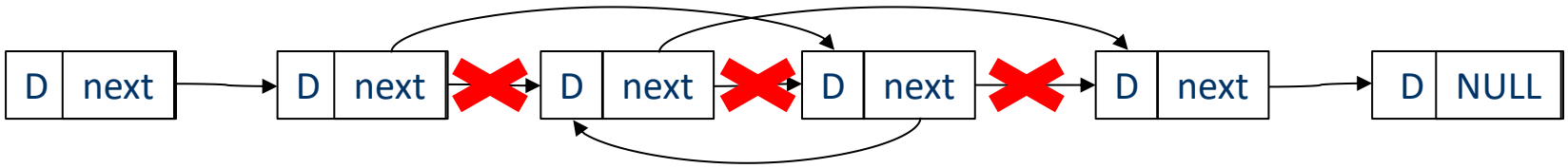
- Παρεμβολή ενός νέου κόμβου



- Διαγραφή κόμβου



- Μετακίνηση κόμβου



- Αναζήτηση κόμβου

Μια απλοϊκή συνάρτηση για δημιουργία μιας Α.Σ.Λ.

```
struct node {
      int  data;
      struct node * next;
} ;
```

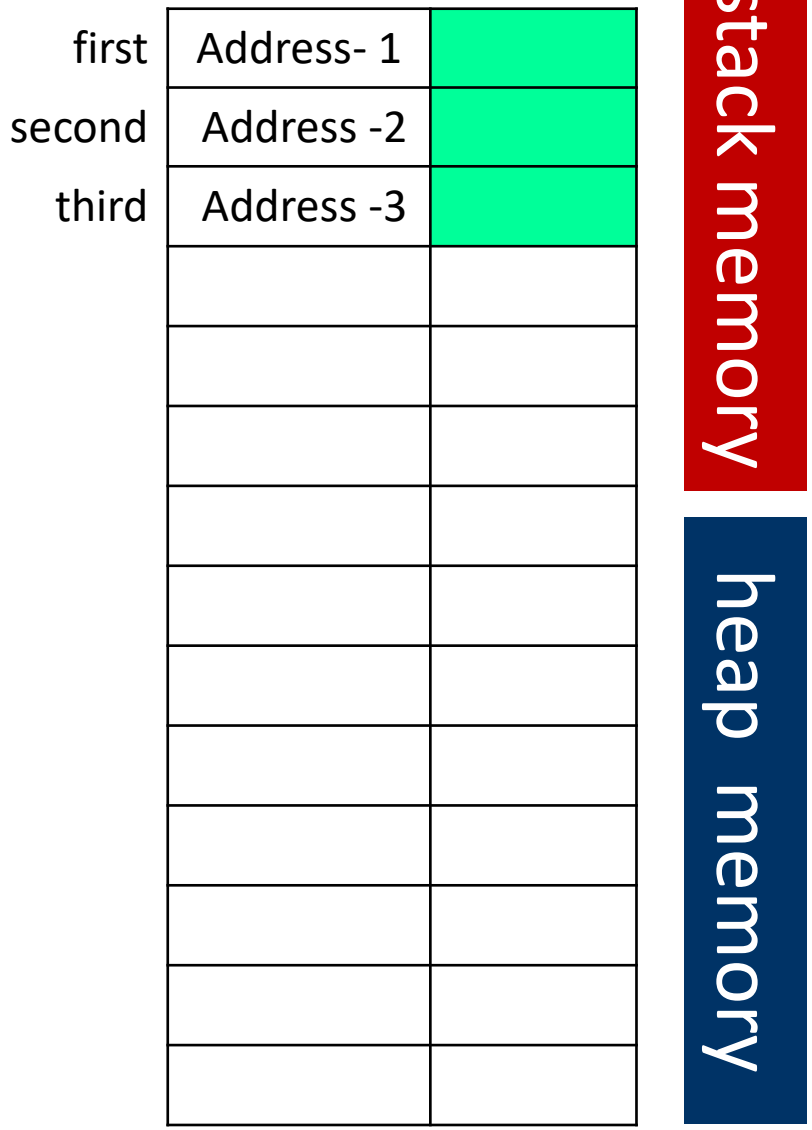
```
struct node* Build123() {
    struct node* first ;
    struct node* second;
    struct node* third ;
    first = (struct node*) malloc(sizeof(struct node));     // μνήμη για 1ο κόμβο
    second = (struct node*) malloc(sizeof(struct node)); // μνήμη για 2ο κόμβο
    third = (struct node*) malloc(sizeof(struct node));     // μνήμη για 3ο κόμβο
    first ->data = 1;           // δίνουμε τιμή στα δεδομένα του 1ου κόμβου
    first ->next = second;  // ο δείκτης του 1ου κόμβου  δείχνει στον 2ο κόμβο
    second->data = 2;        // δίνουμε τιμή στα δεδομένα του 2ου κόμβου
    second->next = third; // ο δείκτης του 2ου κόμβου  δείχνει στον 3ο κόμβο
    third->data = 3;          // δίνουμε τιμή στα δεδομένα του 3ου κόμβου
    third->next = NULL;     // ο δείκτης του 3ου κόμβου  δείχνει στον  NULL
    return first;               //επιστρέφει τη διεύθυνση του 1ου  κόμβου
}
```

Χρειάζεται μόνο στη C++

Δημιουργεί 3 κόμβους, καταχωρίζει τιμές (1,2,3) στις μεταβλητές δεδομένων και τους διασυνδέει ώστε να σχηματίζουν μια Α.Σ.Λ.

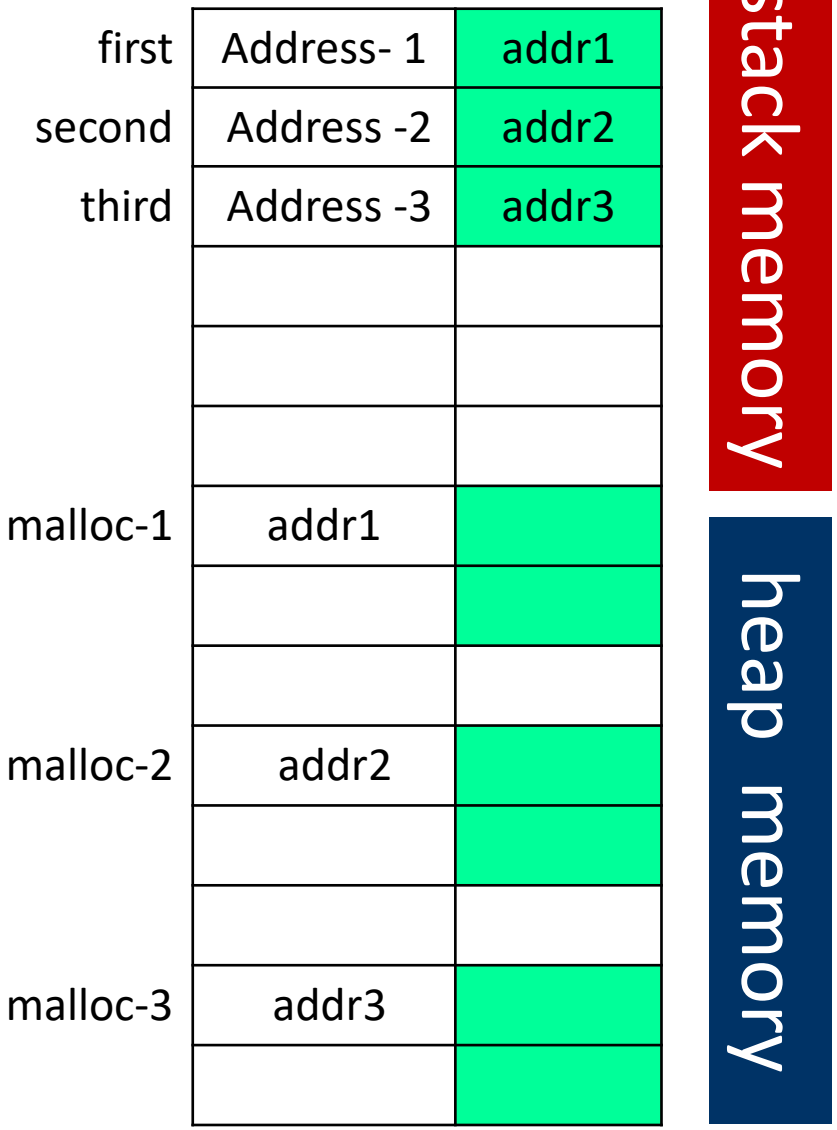
Τι συμβαίνει στη μνήμη όταν εκτελείται η Build123();

```
struct node* Build123() {
    struct node* first ;
    struct node* second;
    struct node* third ;
    first = malloc(sizeof(struct node));
    second = malloc(sizeof(struct node));
    third = malloc(sizeof(struct node));
    first ->data = 1;
    first ->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3;
    third->next = NULL;
    return first;
}
```



Τι συμβαίνει στο σωρό όταν εκτελείται η Build123();

```
struct node* Build123() {
    struct node* first ;
    struct node* second;
    struct node* third ;
    first = malloc(sizeof(struct node));
    second = malloc(sizeof(struct node));
    third = malloc(sizeof(struct node));
    first ->data = 1;
    first ->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3;
    third->next = NULL;
    return first;
}
```



Τι συμβαίνει στο σωρό όταν εκτελείται η Build123();

```
struct node* Build123() {
    struct node* first ;
    struct node* second;
    struct node* third ;
    first = malloc(sizeof(struct node));
    second = malloc(sizeof(struct node));
    third = malloc(sizeof(struct node));
    first ->data = 1;
    first ->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3;
    third->next = NULL;
    return first;
}
```

first	Address- 1	addr1
second	Address -2	addr2
third	Address -3	addr3
malloc-1	addr1	1
		addr2
malloc-2	addr2	2
		addr3
malloc-3	addr3	3
		NULL

stack memory

heap memory

Δημιουργήθηκε όντως μια Α.Σ.Λ με την Build123();

```
void Printlist(struct node* head) {
    struct node* curr;      //προσωρινός κόμβος
    curr=head;              // ο προσωρινός κόμβος δείχνει όπου και ο 1ος κόμβος
    while (curr != NULL){   //μέχρι να φτάσω στον τελευταίο κόμβο
        printf("data:%d\n", curr->data); //εμφάνιση δεδομένων τρέχοντος κόμβου
        curr=curr->next;   // ο προσωρινός κόμβος δείχνει στον επόμενο κόμβο
    }
}
```

συνάρτηση για εμφάνιση των κόμβων μιας Α.Σ.Λ.

```
int Length(struct node* head) {
   struct node* current = head;  //προσωρινός κόμβος που δείχνει όπου και ο 1ος κόμβος
   int count = 0;      //μετρητής κόμβων της Α.Σ.Λ.
   while (current != NULL) {          //μέχρι να φτάσω στον τελευταίο κόμβο
            count++;     //αυξάνω κατά 1 τον μετρητή κόμβων της Α.Σ.Λ.
            current = current->next;  // ο προσωρινός κόμβος δείχνει στον επόμενο κόμβο
   }
  return count;
}
```

συνάρτηση για να βρεθεί ο αριθμός των κόμβων μιας Α.Σ.Λ.

Και το κυρίως πρόγραμμα που καλεί την Build123()

```
struct node {
      int data;
      struct node* next;
};
struct node* Build123();
int Length(struct node* head);
void Printlist(struct node* head);
void Freelistmem(struct node* head);
main() {
      int howmany;
      struct node* head;
      head=Build123();   //δημιουργία Α.Σ.Λ.
      howmany=Length(head);  //Πόσους κόμβους έχει η Α.Σ.Λ.
      printf("\nThere are %d nodes\n", howmany);
      printf("\nThese are:\n");
      Printlist(head);    //εμφάνιση των δεδομένων των κόμβων της Α.Σ.Λ.
      Freelistmem(head);   //αποδέσμευση της μνήμης της Α.Σ.Λ.
}
```

```
//αποδέσμευση της μνήμης της Α.Σ.Λ.
void Freelistmem(struct node* head) {
    struct node *curr, *tmp;
    curr=head;
    while (curr != NULL){
          tmp=curr;
          curr=curr->next;
          free(tmp);
     }
}
```

Εισαγωγή νέου κόμβου στην αρχή υφιστάμενης Α.Σ.Λ.

- Προϋπόθεση:
  - Για τη ευκολότερη διαχείριση της Α.Σ.Λ. χρησιμοποιείται ένας δείκτης που δείχνει στον **πρώτο** (head) κόμβο.
- Διαδικασία:
  - Δέσμευση μνήμης για τον νέο κόμβο:

    struct node* curr; //προσωρινός κόμβος

    curr=(struct node*)malloc(sizeof(struct node));
  - Δίνουμε τιμές στις μεταβλητές δεδομένων του νέου κόμβου:

    curr->data=5; //δίνω μια τιμή στη μεταβλητή πχ 5
  - Καταχωρίζουμε στο δείκτη του νέου (προσωρινού) κόμβου την διεύθυνση του 1ου κόμβου στη Α.Σ.Λ.:

    curr->next=head; //ο επόμενος κόμβος είναι ο υφιστάμενος 1ος
  - Αλλάζουμε το περιεχόμενο του δείκτη διαχειριστή (head):

    head=curr; //τώρα ο head δείχνει στο νέο 1ο κόμβο

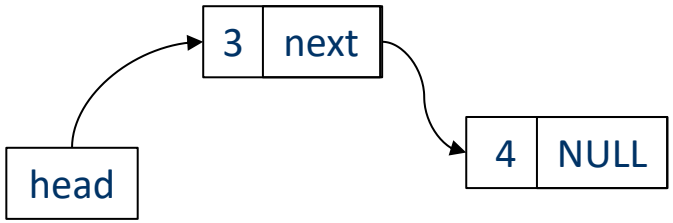
```
struct node {
    int  data;
    struct node * next;
} ;
```

Εισαγωγή νέου κόμβου στην αρχή υφιστάμενης Α.Σ.Λ.

head != NULL

Η δομή:

```
struct node {
    int  data;
    struct node * next;
} ;
```



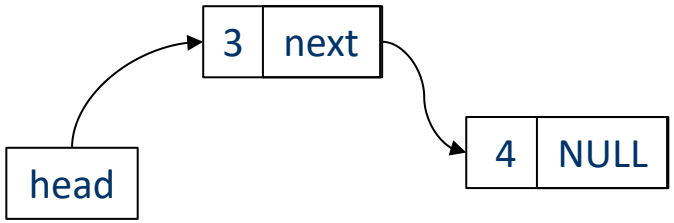
head	haddr	add1
	addr1	3
		addr2
	addr2	4
		NULL

Εισαγωγή νέου κόμβου στην αρχή υφιστάμενης Α.Σ.Λ.

head != NULL

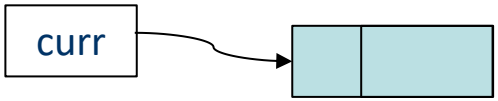
Η δομή:

```
struct node {
     int  data;
     struct node * next;
} ;
```



stuct node *curr;

curr= (struct node *) malloc(sizeof (sruct node));



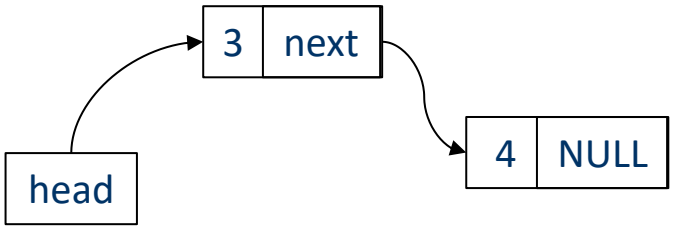
head	haddr	add1
	addr1	3
		addr2
	addr2	4
		NULL
new	naddr	
curr	caddr	naddr

Εισαγωγή νέου κόμβου στην αρχή υφιστάμενης Α.Σ.Λ.

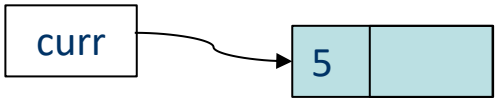
head != NULL

Η δομή:

```
struct node {
    int  data;
    struct node * next;
} ;
```



curr->data= 5;



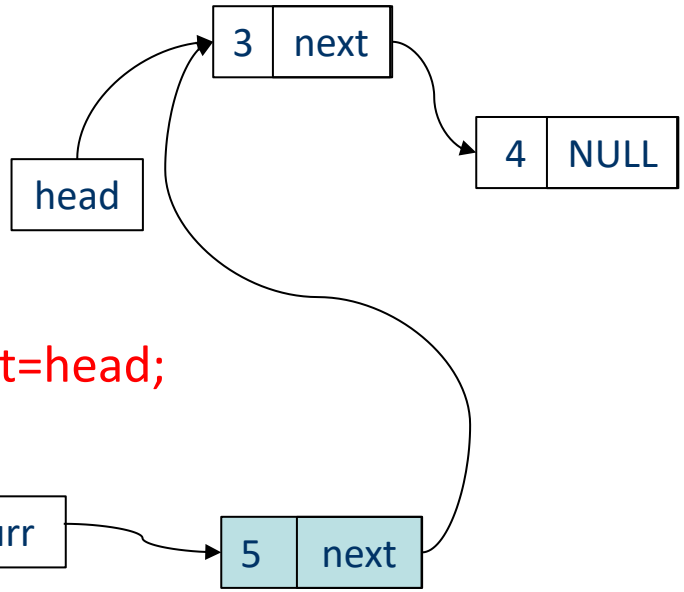
head	haddr	add1
	addr1	3
		addr2
	addr2	4
		NULL
new	naddr	5
curr	caddr	naddr

Εισαγωγή νέου κόμβου στην αρχή υφιστάμενης Α.Σ.Λ.

head != NULL

Η δομή:

```
struct node {
    int  data;
    struct node * next;
} ;
```



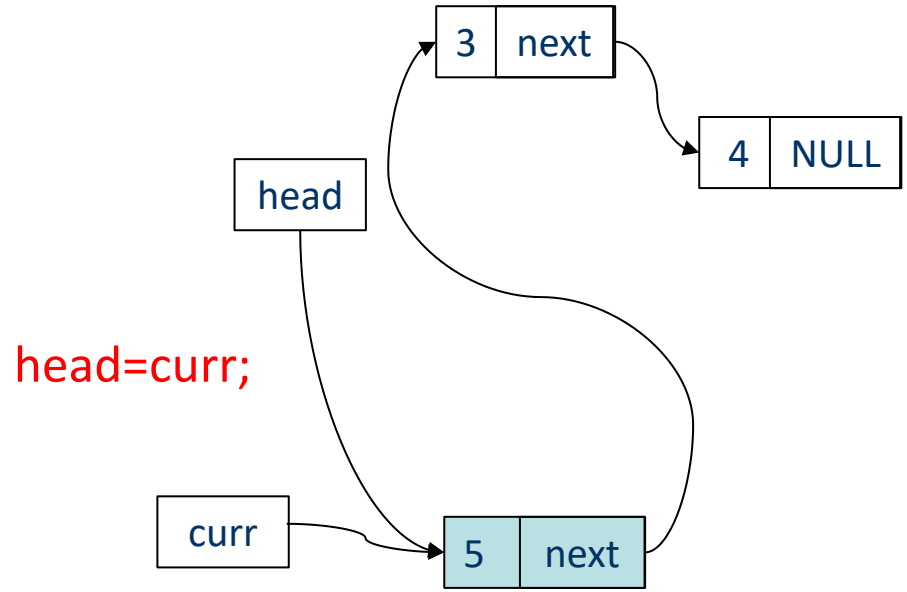
head	haddr	add1
	addr1	3
		addr2
	addr2	4
		NULL
new	naddr	5
		addr1
curr	caddr	naddr

Εισαγωγή νέου κόμβου στην αρχή υφιστάμενης Α.Σ.Λ.

head != NULL

Η δομή:

```
struct node {
    int  data;
    struct node * next;
} ;
```



head	haddr	naddr
	addr1	3
		addr2
	addr2	4
		NULL
new	naddr	5
		addr1
curr	caddr	naddr

Συνάρτηση για εισαγωγή νέου κόμβου σε Α.Σ.Λ. με head

```
//δήλωση της δομής
typedef struct node {
        int data;
       struct node *next;
}    sll_node;
sll_node *head; //δημιουργία head
```

```
//κλήση της συνάρτησης από το main()
     scanf("%d",&x); fflush(stdin);
     add2list(x);
```

```
// ορισμός της συνάρτησης για εισαγωγή νέου κόμβου
void add2list (int d) //d η τιμή για τη μεταβλητή data του κόμβου
{
        sll_node *curr;
        curr=(struct node*)malloc(sizeof(struct node));
        curr->data=d;
        curr->next=head;
        head=curr;
}
```

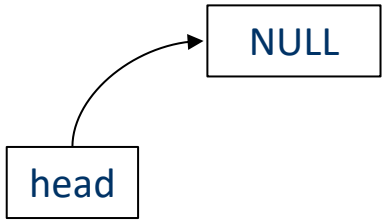
Δουλεύει η συνάρτηση όταν η Α.Σ.Λ είναι κενή -δηλ. έχουμε head=NULL?

Εισαγωγή κόμβου στην αρχή κενής Α.Σ.Λ.

head == NULL

Η δομή:

```
struct node {
    int  data;
    struct node * next;
} ;
```



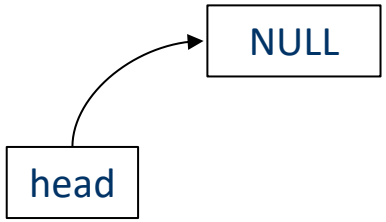
head	haddr	NULL

Εισαγωγή κόμβου στην αρχή κενής Α.Σ.Λ.

head == NULL

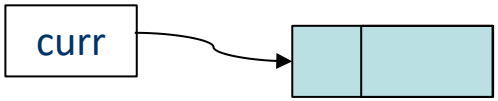
Η δομή:

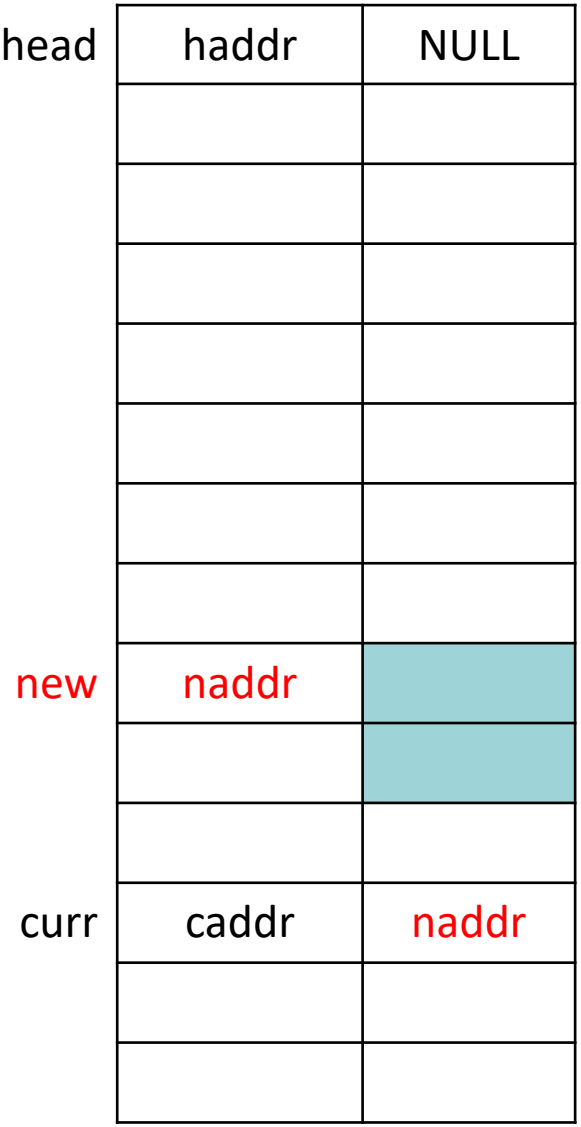
```
struct node {
     int  data;
     struct node * next;
} ;
```



stuct node *curr;

curr= (struct node *) malloc(sizeof (sruct node));



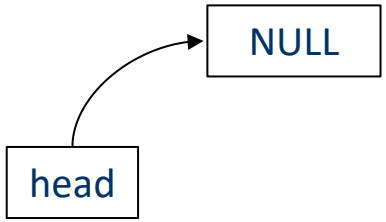


Εισαγωγή κόμβου στην αρχή κενής Α.Σ.Λ.

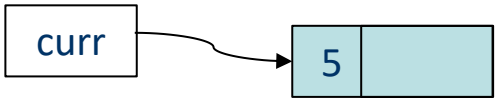
head == NULL

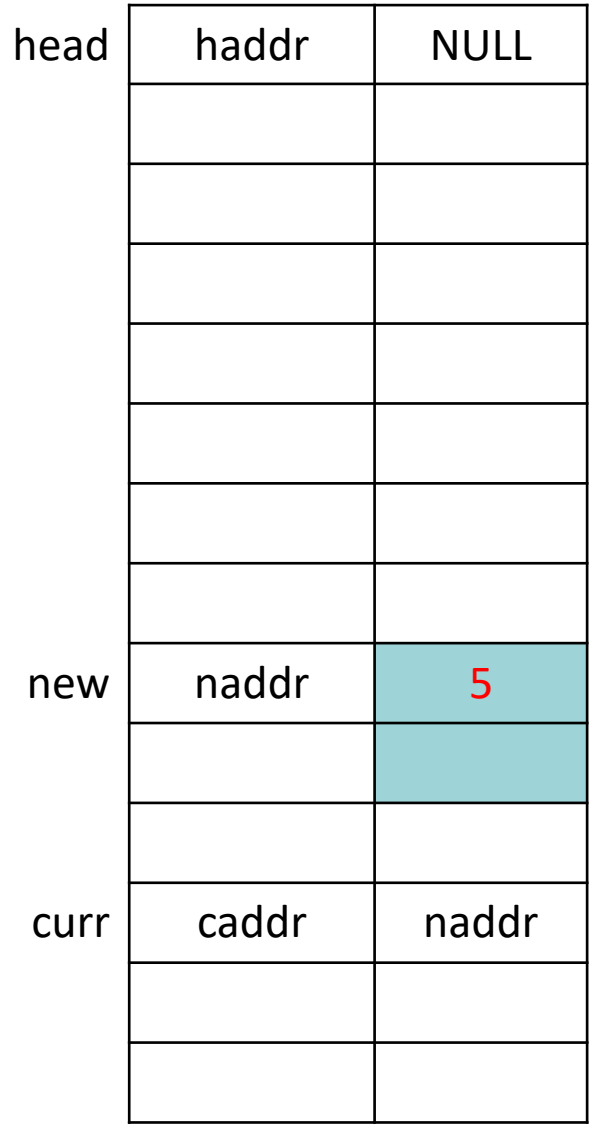
Η δομή:

```
struct node {
     int  data;
     struct node * next;
} ;
```



curr->data= 5;



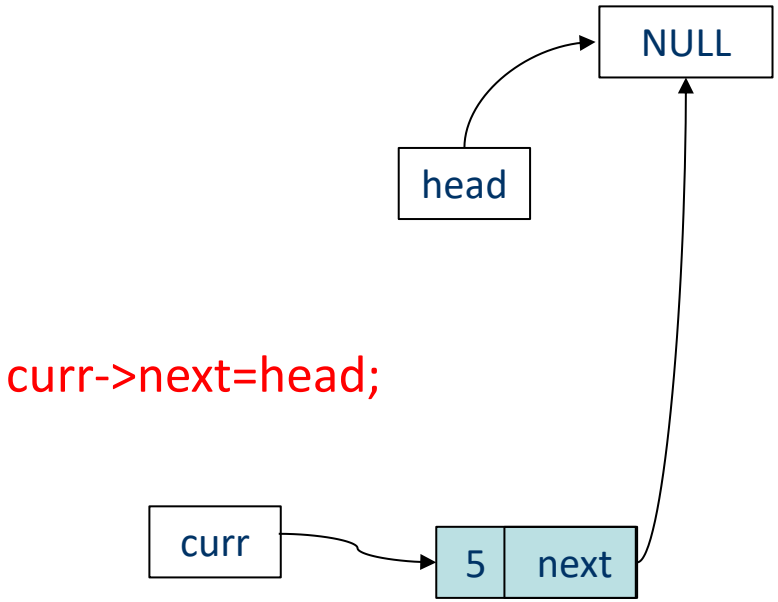


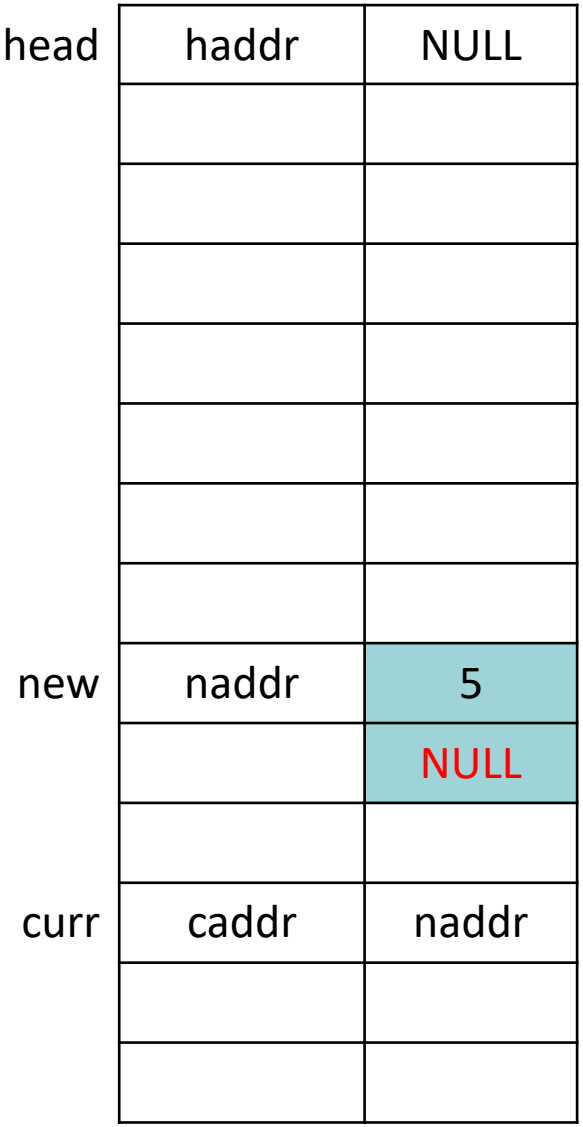
Εισαγωγή κόμβου στην αρχή κενής Α.Σ.Λ.

head == NULL

Η δομή:

```
struct node {
    int  data;
    struct node * next;
} ;
```



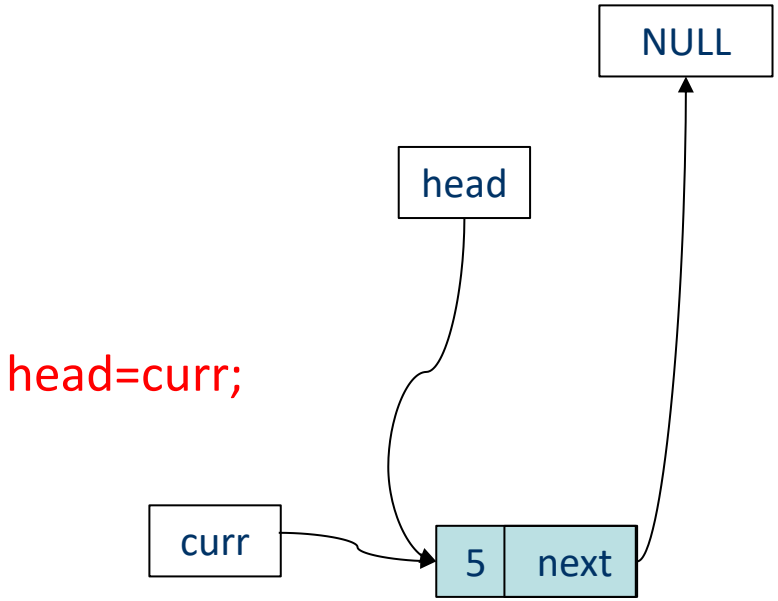


Εισαγωγή κόμβου στην αρχή κενής Α.Σ.Λ.

head == NULL

Η δομή:

```
struct node {
    int  data;
    struct node * next;
} ;
```



head	haddr	naddr
new	naddr	5
		NULL
curr	caddr	naddr

Δημιουργία Α.Σ.Λ. με head – το πλήρες πρόγραμμα

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
     int data;
     struct node *next;
} sll_node;
sll_node *head; // head node
void add2list(int);
void printlist();
void freelistmem();
main() {
   int x;
   head=NULL;
   while (1) {
      printf("Enter number to insert : ");
      scanf("%d",&x);fflush(stdin);
      if (x<0) break;  //stops the program
      add2list(x);
   }
   printlist();
   freelistmem();
}
```

```c
void printlist() { //εμφάνιση Α.Σ.Λ
    sll_node *curr;
    curr=head;
    while (curr != NULL){
         printf("data:%d\n", curr->data);
         curr=curr->next;
    }
}
```

```c
void freelistmem() { //αποδέσμευση μνήμης
   sll_node *curr, *tmp;
   curr=head;
   while (curr != NULL){
         tmp=curr;
         curr=curr->next;
         free(tmp);
  }
}
```

Πρόγραμμα για τη δημιουργία μιας Α.Σ.Λ.
Κάθε νέος κόμβος εισάγεται στη αρχή.
Σταματά όταν δοθεί αρνητικός αριθμός

Υλοποίηση μηχανισμού στοίβας (stack)

- Μια στοίβα είναι ένας μηχανισμός LIFO (Last In, First Out) :

  (το τελευταίο στοιχείο που μπαίνει στη στοίβα είναι και το πρώτο που βγαίνει)

- Μπορούμε να χρησιμοποιήσουμε τη συνάρτηση για την εισαγωγή νέου κόμβου στην αρχή μιας Α.Σ.Λ. που εξετάσαμε προηγουμένως (Last In).

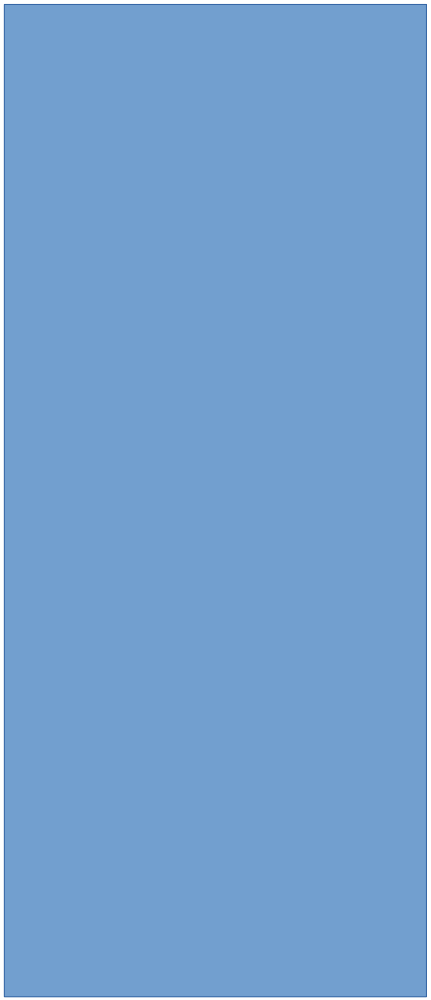
  Στη ορολογία της στοίβας αυτή η διαδικασία ονομάζεται **push**.

- Θα εξετάσουμε μια άλλη συνάρτηση για να εξάγουμε το πρώτο στοιχείο της Α.Σ.Λ. (First Out). Πρόκειται για τον τελευταίο (πιο πρόσφατο) νέο κόμβο.

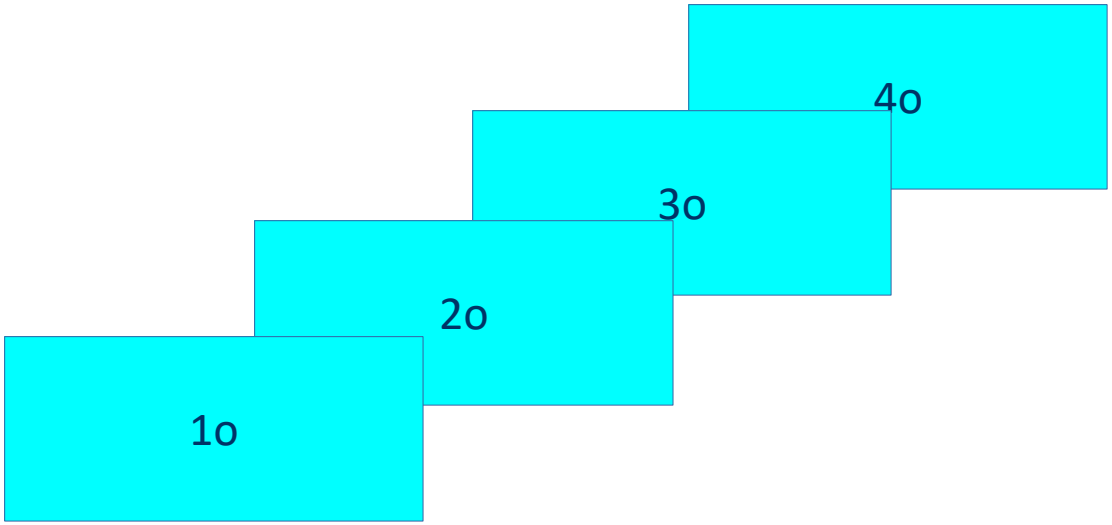
  Στη ορολογία της στοίβας αυτή η διαδικασία ονομάζεται **pop**.

- Για τη ολοκλήρωση του προγράμματος θα δημιουργήσουμε κάποιες ακόμα χρήσιμες συναρτήσεις:
  - για την εμφάνιση όλων των στοιχείων της στοίβας και
  - για την εμφάνιση του 1^{ου} στοιχείου τη στοίβας.

Η στοίβα ως μηχανισμός LIFO

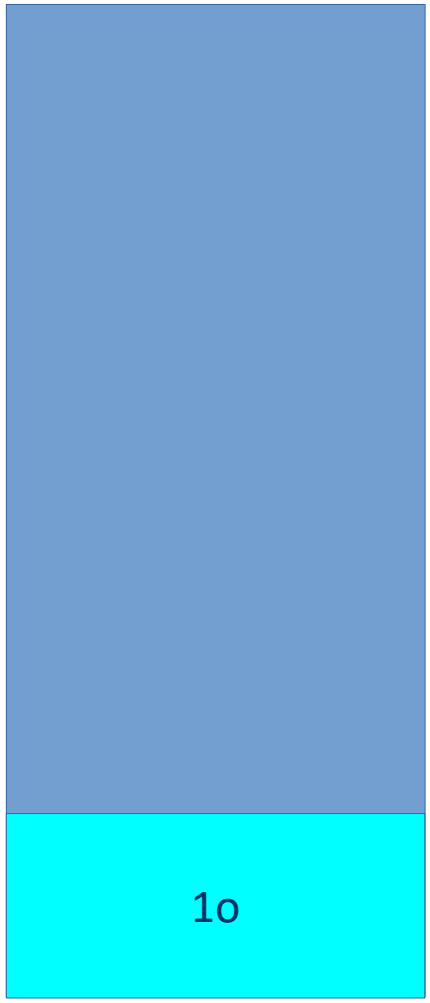


Η στοίβα είναι άδεια

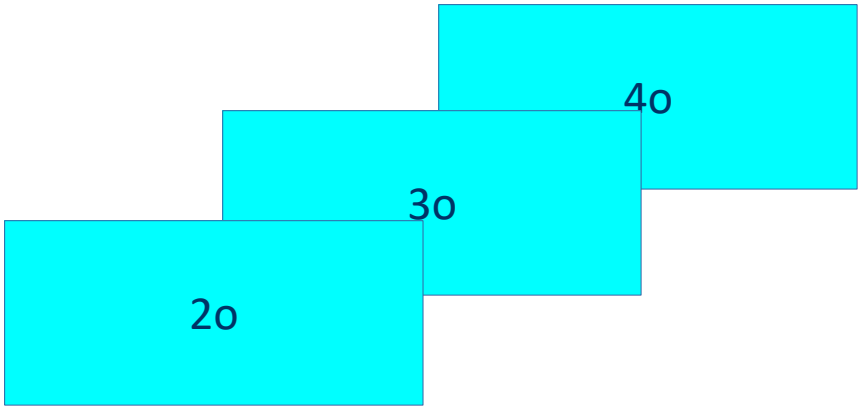


ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



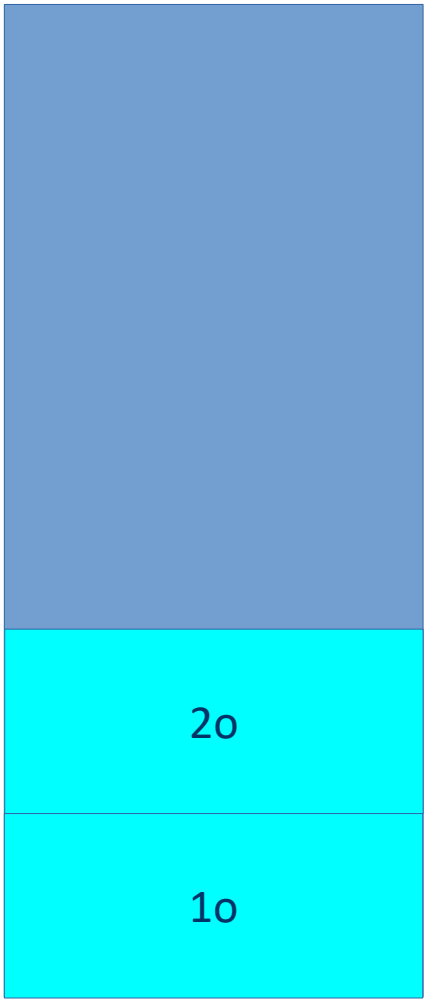
1 στοιχείο στη στοίβα



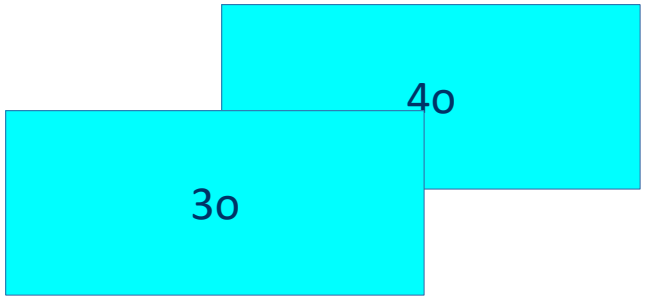
Ώθηση (push)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



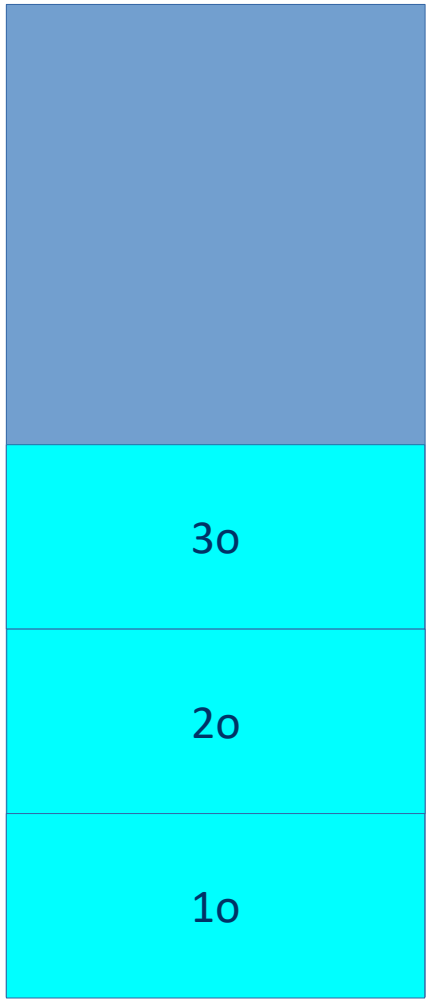
2 στοιχεία στη στοίβα



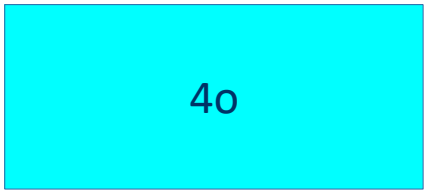
Ώθηση (push)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



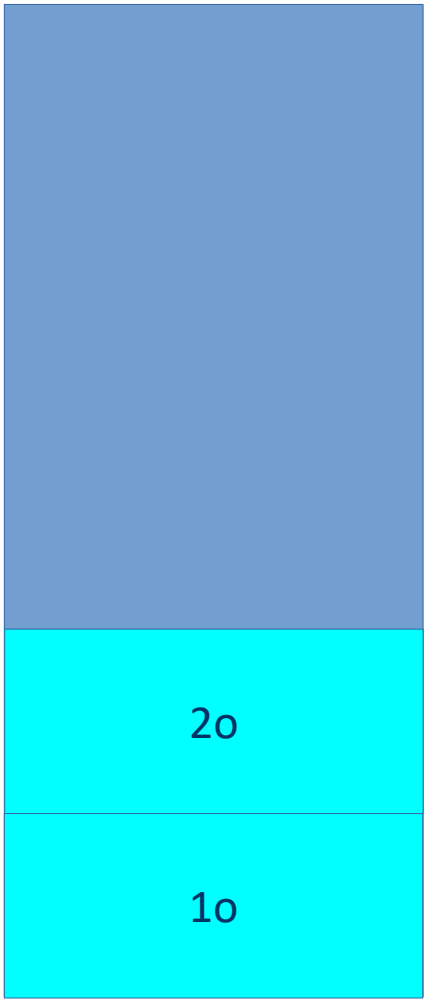
3 στοιχεία στη στοίβα



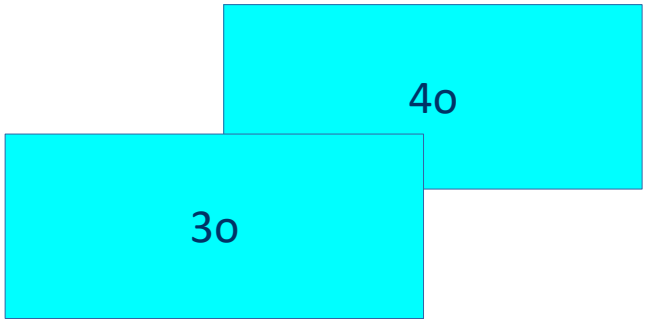
Ώθηση (push)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



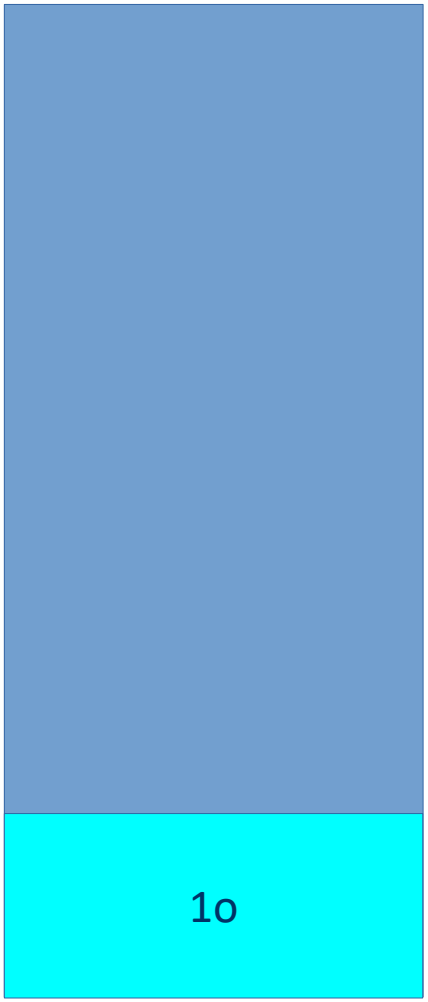
2 στοιχεία στη στοίβα



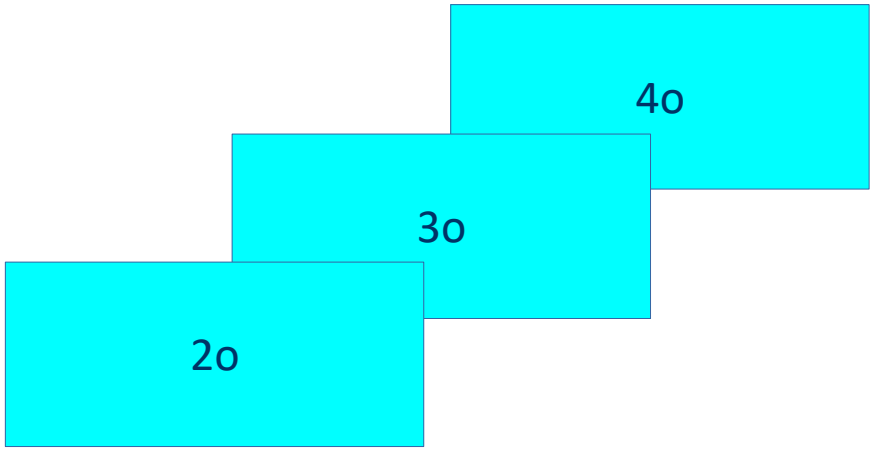
απώθηση (pop)

ΣΤΟΙΒΑ (stack)

Η στοίβα ως μηχανισμός LIFO



1 στοιχείο στη στοίβα



απώθηση (pop)

ΣΤΟΙΒΑ (stack)

Υλοποίηση μηχανισμού στοίβας – η συνάρτηση push()

```
// ορισμός της συνάρτησης για εισαγωγή νέου κόμβου
//d η τιμή για τη μεταβλητή data του κόμβου
int push(int d) {
        stack_node *tmp;
        tmp = (struct node*)malloc(sizeof(struct node));
        if (tmp==NULL) return 0; //έλεγχος για επαρκή μνήμη
        tmp->data=d;
        tmp->next = top;
        top =tmp;
       return 1;        //εισήχθη επιτυχώς
}
```

Η δομή:

```
typedef struct node {
                int data;
                struct node *next;
} stack_node;
stack_node *top = NULL;
```

Η συνάρτηση **push()** είναι ίδια με τη συνάρτηση **add2list().** Μόνο που:

- Αντί για **curr** έχουμε **tmp**
- Αντί για **head** έχουμε **top**

```
void add2list (int d) {
        sll_node *curr;
        curr=(struct node*)malloc(sizeof(struct node));
        curr->data=d;
        curr->next=head;
        head=curr;
}
```

Υλοποίηση μηχανισμού στοίβας – η συνάρτηση pop()

```
void pop() {
      stack_node *temp;
      int d;
      if (isempty()){  //Έλεγχος αν η στοίβα είναι άδεια
            puts("Empty stack\n");
            return;
       }
     printf("The number:%d has been removed!\n",top->data);
     temp = top->next;  //ο temp δείχνει στον 2ο κόμβο
     free(top);  //αποδέσμευση μνήμης
     top = temp;  // ο top δείχνει στον 2ο κόμβο
}
```

Η δομή:

```
typedef struct node {
            int data;
            struct node *next;
} stack_node;
stack_node *top = NULL;
```

```
//Έλεγχος αν η στοίβα είναι άδεια
int isempty() {
      if (top==NULL)  return 1;
      else return 0;
}
```

Υλοποίηση μηχανισμού στοίβας – άλλες συναρτήσεις

```
//Συνάρτηση για εμφάνιση της στοίβας
void displaystack() {
      stack_node *curr;
      curr=top;
      while (curr != NULL){
            printf(" %d\n", curr->data);
            curr=curr->next;
       }
}
```

Η δομή:

```
typedef struct node {
                int data;
                struct node *next;
} stack_node;
stack_node *top = NULL;
```

```
//Αποδέσμευση μνήμης στοίβας
void freestackmem()  {
      stack_node *curr, *tmp;
      curr=top;
      while (curr != NULL){
            tmp=curr;
            curr=curr->next;
            free(tmp);
       }
}
```

```
//Εμφανίζει τον 1ο κόμβο της στοίβας
void showtop() {
      if (isempty())
            puts("Empty stack\n");
      else
            printf("Now top is:%d\n", top->data);
}
```

Υλοποίηση μηχανισμού στοίβας – όλο το πρόγραμμα

```
 #include <stdio.h>
#include <stdlib.h>
//Define stack
typedef struct node {
            int data;
            struct node *next;
} stack_node;
// Initialize stack
stack_node *top = NULL;
// Used functions
void pop();
int push(int d);
int isempty();
void showtop();
void displaystack();
void freestackmem();
```

```
int main(void) {
 char ch;
 int num;
 while (1) {
    system("cls");
    printf("\n0: Exit\n1: Push\n2: Pop\n3: Show top\n4: Display stack\n");
    printf("Your choice:");
    scanf("%c",&ch);fflush(stdin);
    switch(ch) {
        case '0':  freestackmem();  exit(0);
        case '1':  printf("Number to push into:");
                   scanf("%d",&num);fflush(stdin);
                   if (push(num)==0)  puts("No more memory!\n");
                   break;
       case '2': pop(); getchar(); break;
       case '3': showtop(); getchar();  break;
       case '4': displaystack(); getchar(); break;
       default:  puts("Please try again"); break;
    }
   }
   system("pause");
}
```

Εισαγωγή νέου κόμβου στο τέλος υφιστάμενης Α.Σ.Λ. (με head & tail)

- Προϋπόθεση:
  - Για τη ευκολότερη διαχείριση της Α.Σ.Λ. χρησιμοποιούνται 2 δείκτες , ένας για την αρχή (head) και ένας για το τέλος της λίστας (tail).
- Διαδικασία:
  - Δέσμευση μνήμης για τον νέο κόμβο:

    curr=(struct node*)malloc(sizeof(struct node));
  - Ο δείκτης του τελευταίου κόμβου δείχνει στον νέο κόμβο:

    tail->next=curr;
  - Δίνουμε τιμές στις μεταβλητές δεδομένων του νέου κόμβου:

    curr->data=5;
  - Ο νέος κόμβος δείχνει στο NULL (θα είναι ο τελευταίος):

    curr->next=NULL;
  - Αλλάζουμε το περιεχόμενο του δείκτη διαχειριστή (tail):

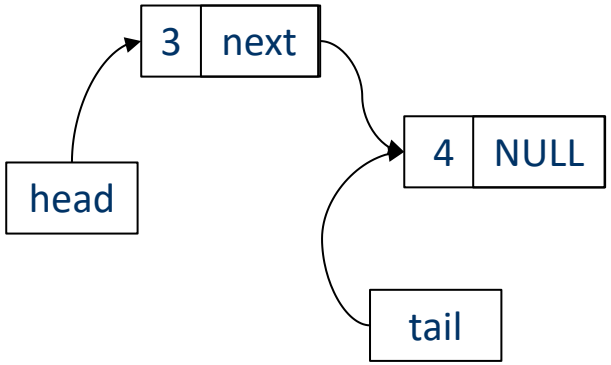
    tail = curr; //τώρα ο tail δείχνει στον τελευταίο κόμβο

Παρατήρηση: *Ο δείκτης στον πρώτο κόμβο (head) δεν αλλάζει.*

Εισαγωγή νέου κόμβου στο τέλος υφιστάμενης Α.Σ.Λ. (με head & tail)

head != NULL

```
struct node {
    unsigned int data
   struct node *next
}
```

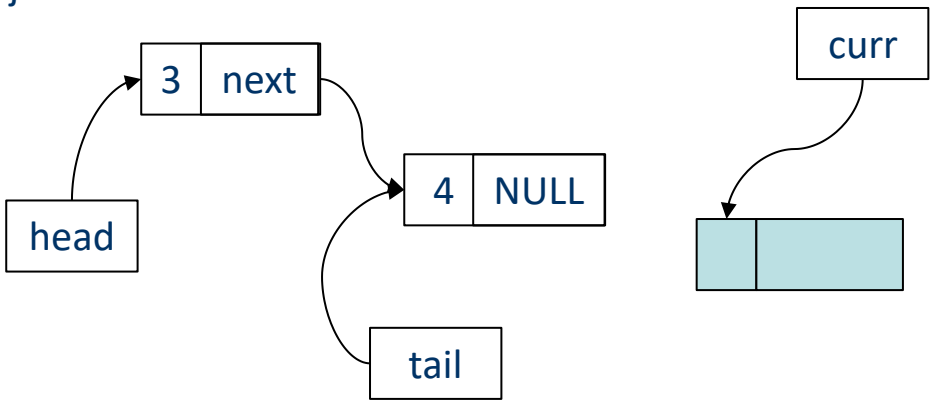


head	haddr	add1
tail	taddr	add2
	addr1	3
		addr2
	addr2	4
		NULL

Εισαγωγή νέου κόμβου στο τέλος υφιστάμενης Α.Σ.Λ. (με head & tail)

head != NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



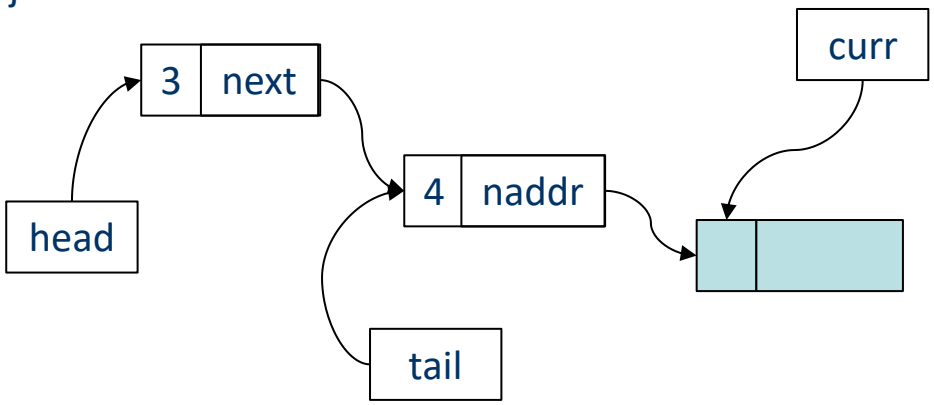
curr= (struct node *) malloc(sizeof (sruct node));

head	haddr	add1
tail	taddr	add2
	addr1	3
		addr2
	addr2	4
		NULL
new	naddr	
curr	caddr	naddr

Εισαγωγή νέου κόμβου στο τέλος υφιστάμενης Α.Σ.Λ. (με head & tail)

head != NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



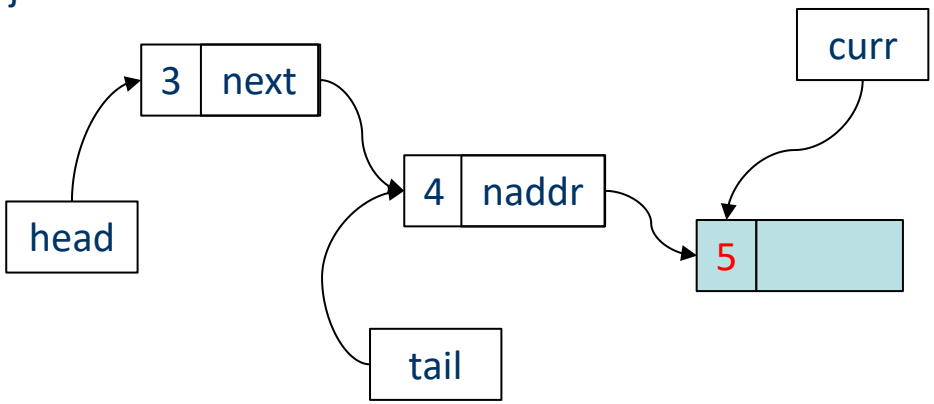
tail->next=curr;

head	haddr	add1
tail	taddr	add2
	addr1	3
		addr2
	addr2	4
		naddr
new	naddr	
curr	caddr	naddr

Εισαγωγή νέου κόμβου στο τέλος υφιστάμενης Α.Σ.Λ. (με head & tail)

head != NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



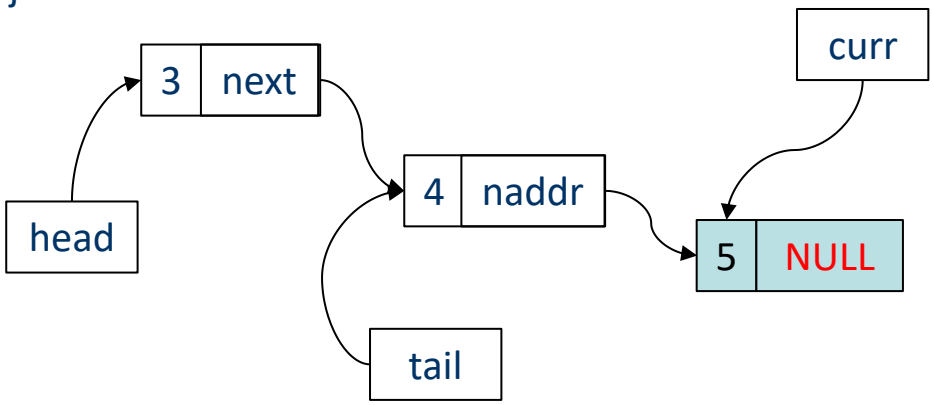
curr->data=5;

head	haddr	add1
tail	taddr	add2
	addr1	3
		addr2
	addr2	4
		naddr
new	naddr	5
curr	caddr	naddr

Εισαγωγή νέου κόμβου στο τέλος υφιστάμενης Α.Σ.Λ. (με head & tail)

head != NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



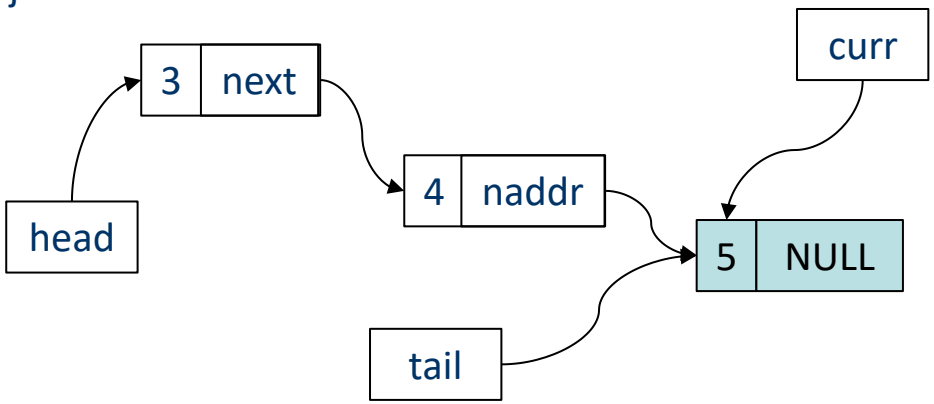
curr->next=NULL;

head	haddr	add1
tail	taddr	add2
	addr1	3
		addr2
	addr2	4
		naddr
new	naddr	5
		NULL
curr	caddr	naddr

Εισαγωγή νέου κόμβου στο τέλος υφιστάμενης Α.Σ.Λ. (με head & tail)

head != NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



tail = curr;

head	haddr	add1
tail	taddr	naddr
	addr1	3
		addr2
	addr2	4
		naddr
new	naddr	5
		NULL
curr	caddr	naddr

Εισαγωγή νέου κόμβου σε κενή Α.Σ.Λ. με head & tail

- Προϋπόθεση:
  - Για τη ευκολότερη διαχείριση της Α.Σ.Λ. χρησιμοποιούνται 2 δείκτες , ένας για την αρχή (head) και ένας για το τέλος της λίστας (tail).
- Διαδικασία:
  - Δέσμευση μνήμης για τον νέο κόμβο:

    curr=(struct node*)malloc(sizeof(struct node));
  - Ο δείκτης στον πρώτο κόμβο δείχνει στο νέο κόμβο:

    head = curr;
  - Δίνουμε τιμές στις μεταβλητές δεδομένων του νέου κόμβου:

    curr->data=5;
  - Ο νέος κόμβος δείχνει στο NULL (θα είναι ο τελευταίος):

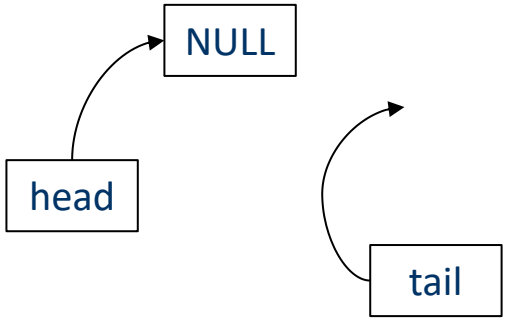
    curr->next=NULL;
  - Αλλάζουμε το περιεχόμενο του δείκτη διαχειριστή (tail):

    tail = curr; //τώρα ο tail δείχνει στον τελευταίο κόμβο

Εισαγωγή νέου κόμβου σε κενή Α.Σ.Λ. με head & tail

head == NULL

```
struct node {
    unsigned int data
   struct node *next
}
```

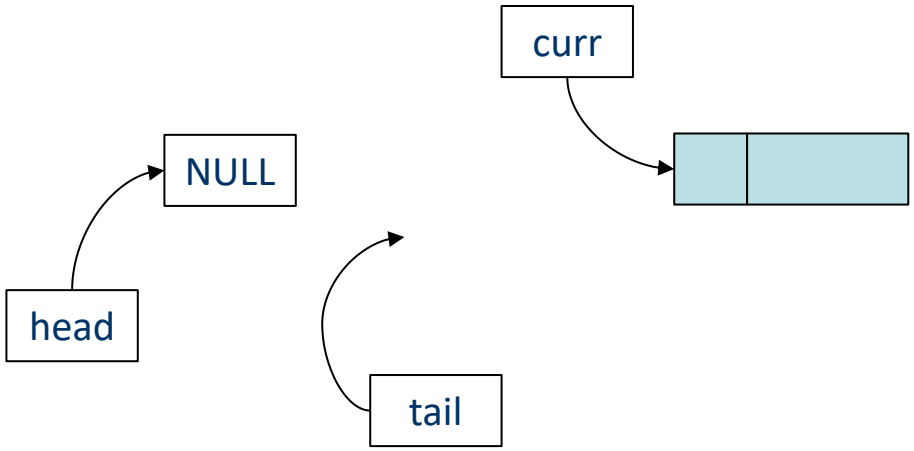


head	haddr	NULL
tail	taddr	

Εισαγωγή νέου κόμβου σε κενή Α.Σ.Λ. με head & tail

head == NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



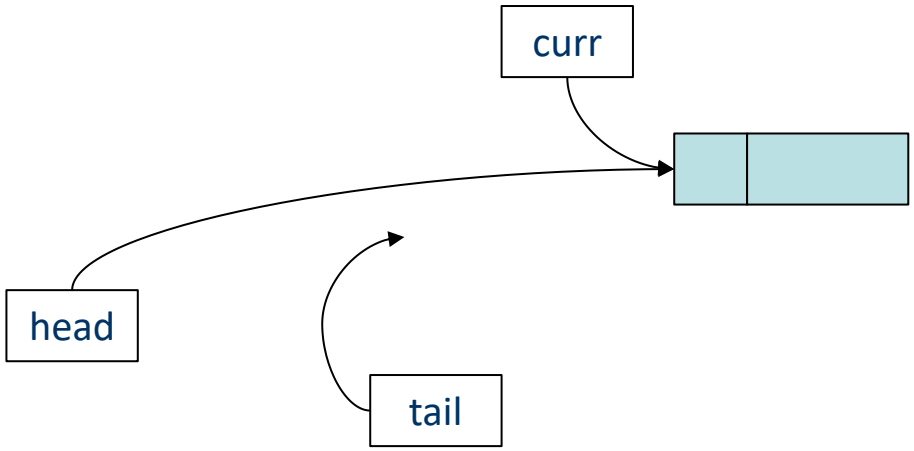
curr= (struct node *) malloc(sizeof (sruct node);

head	haddr	NULL
tail	taddr	
new	naddr	
curr	caddr	naddr

Εισαγωγή νέου κόμβου σε κενή Α.Σ.Λ. με head & tail

head == NULL

```
struct node {
    unsigned int data
   struct node *next
}
```

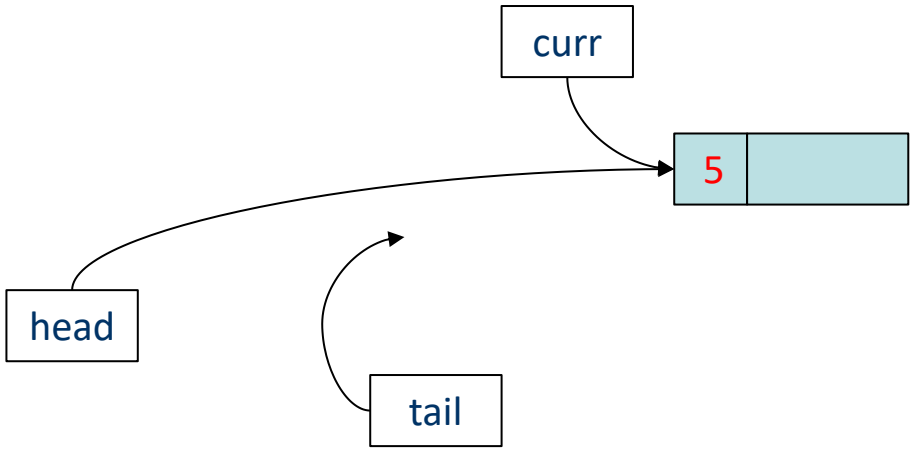


head = curr;

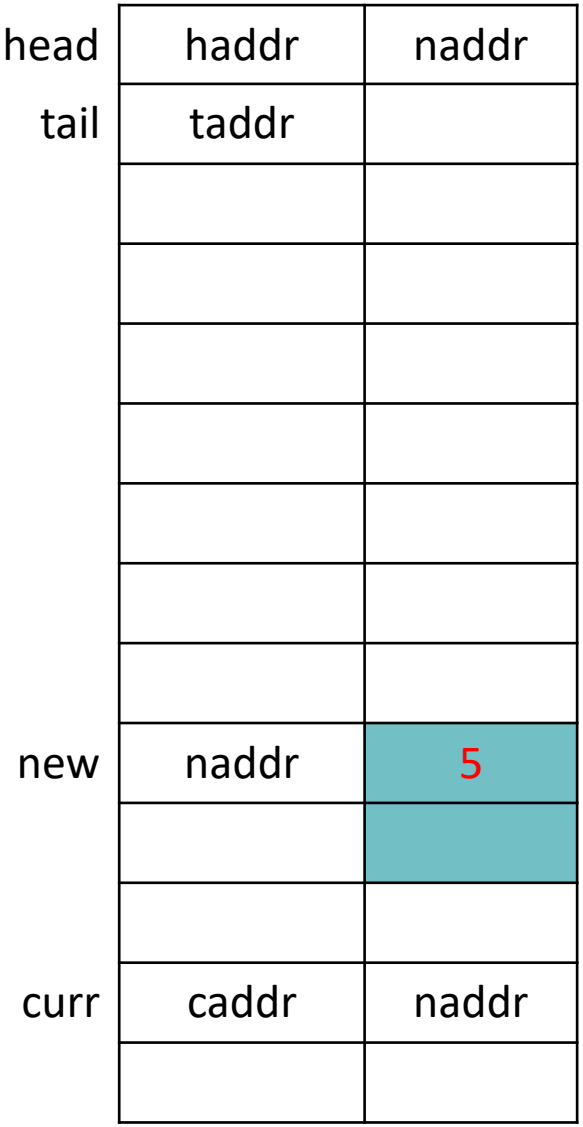
Εισαγωγή νέου κόμβου σε κενή Α.Σ.Λ. με head & tail

head == NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



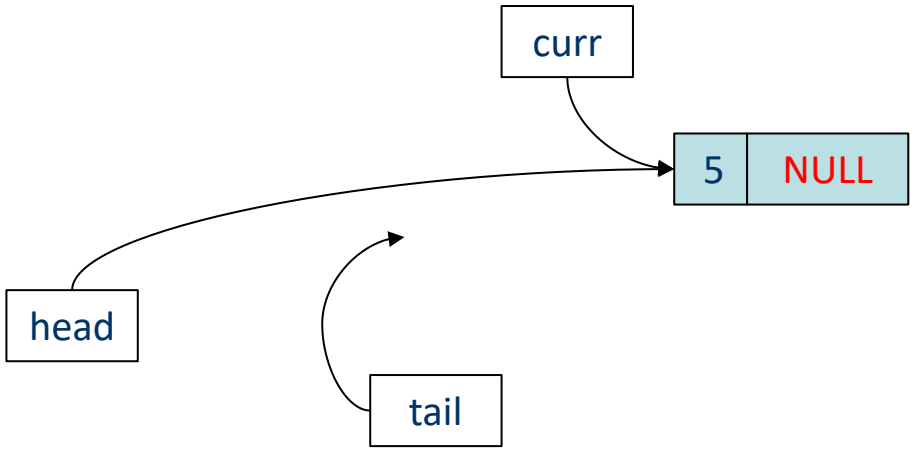
curr ->data= 5;



Εισαγωγή νέου κόμβου σε κενή Α.Σ.Λ. με head & tail

head == NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



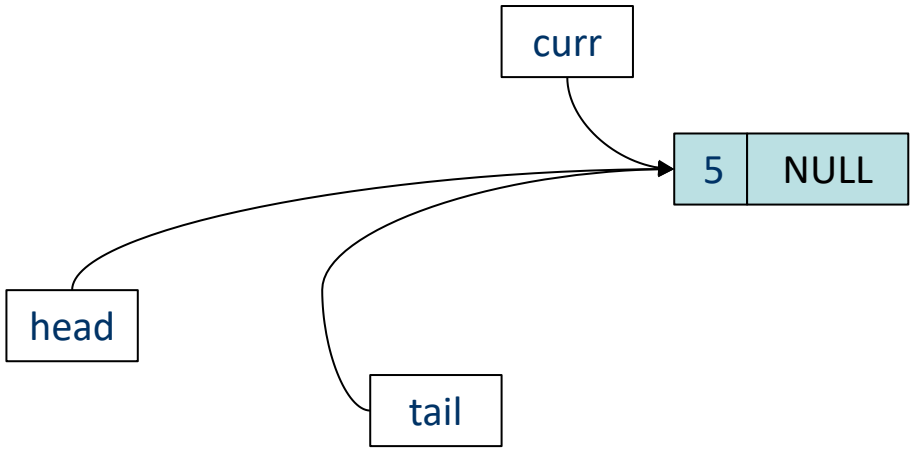
curr ->next= NULL;

head	haddr	naddr
tail	taddr	
new	naddr	5
		NULL
curr	caddr	naddr

Εισαγωγή νέου κόμβου σε κενή Α.Σ.Λ. με head & tail

head == NULL

```
struct node {
    unsigned int data
   struct node *next
}
```



tail = curr;

head	haddr	naddr
tail	taddr	
new	naddr	1
		NULL
curr	caddr	naddr

Η νέα συνάρτηση add2list()

```
// ορισμός της συνάρτησης για εισαγωγή νέου κόμβου
void add2list (int d) //d η τιμή για τη μεταβλητή data του κόμβου
{
        sll_node *curr;
        curr=(struct node*)malloc(sizeof(struct node));
        if (head ==NULL) head=curr;
        else tail->next=curr;
        curr->data=d;
        curr->next=NULL;
        tail=curr;
}
```