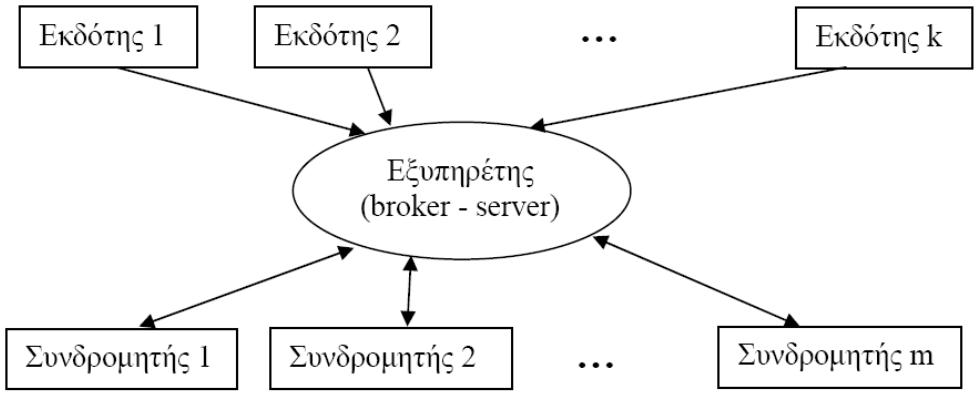
HY240

Φροντιστήριο Project

HY240

Ενότητα 1η: Περιγραφή Project

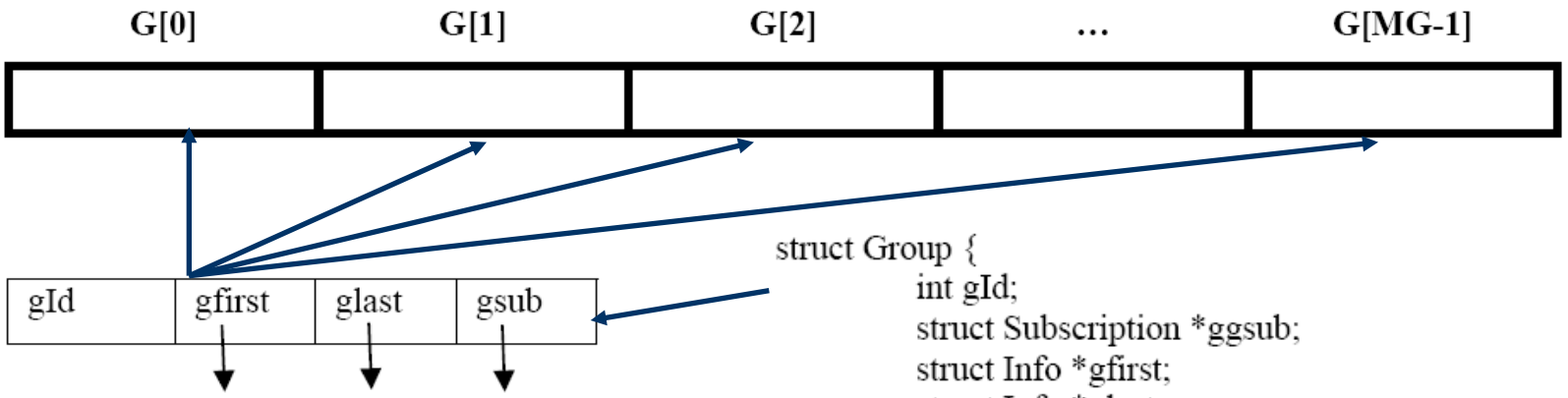
Σχηματική Αναπαράσταση



- Οι πληροφορίες αυτές κατηγοριοποιούνται σε ομάδες (groups) βάσει του περιεχομένου τους ή κάποιων λέξεων κλειδιών που παρέχονται από τους εκδότες.

- οι εκδότες (publishers) είναι παροχείς πληροφοριών
- οι συνδρομητές είναι οι καταναλωτές των πληροφοριών αυτών
- ο κεντρικός εξυπηρέτης (broker-server) λειτουργεί σαν ενδιάμεσος κόμβος μεταξύ των εκδοτών και των συνδρομητών

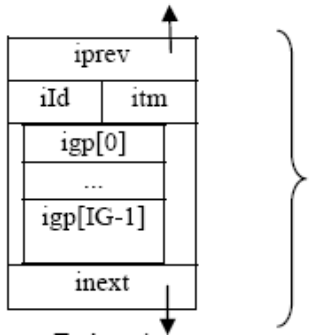
Groups: Κατηγοριοποιηση πληροφοριών



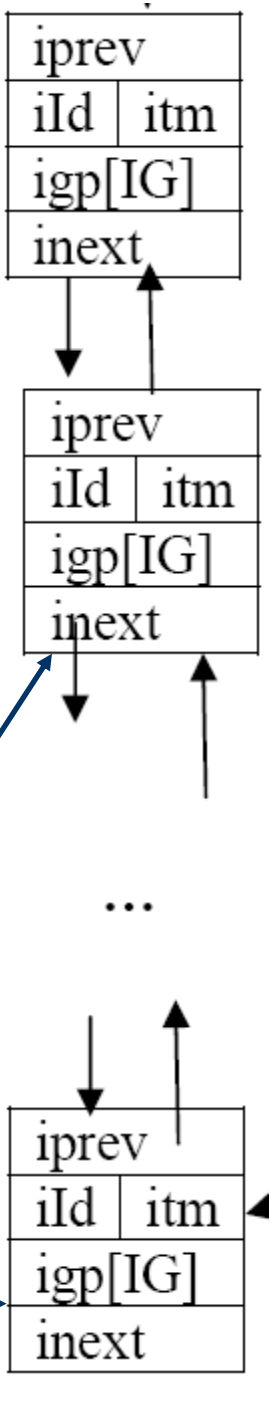
- gId: το αναγνωριστικό της ομάδας.
- gfirst και glast: Δυο δείκτες, οι οποίοι δείχνουν στο πρώτο και στο τελευταίο στοιχείο, αντίστοιχα, μιας διπλά συνδεδεμένης, ταξινομημένης, λίστας, η οποία ονομάζεται *λίστα πληροφοριών*. Η λίστα αυτή περιέχει τις πληροφορίες που συσχετίζονται με αυτή την ομάδα.
- gsub: Ένας δείκτης στο πρώτο στοιχείο μιας απλά συνδεδεμένης λίστας που αποθηκεύει πληροφορίες για τις συνδρομές σε αυτή την ομάδα και ονομάζεται *λίστα συνδρομών*.

Groups: Λίστα Πληροφοριών Ομάδας

- iId: Ακέραιος που αποτελεί το αναγνωριστικό της πληροφορίας.
- ο Έναν ακέραιο itm που αποθηκεύει την «χρονική στιγμή» στην οποία η πληροφορία αυτή καταφθάνει στο σύστημα. Η λίστα πληροφοριών της εκάστοτε ομάδας είναι ταξινομημένη βάσει του πεδίου itm των εγγραφών της.
- ο Έναν πίνακα igp[MG] των MG (Maximum_Groups) θέσεων που περιγράφει τα groups πληροφοριών στα οποία ανήκει η πληροφορία. Εάν ισχύει igp[i] == 1, 0≤i≤MG-1, τότε η πληροφορία έχει συσχετισθεί (ή ανήκει) στην ομάδα πληροφοριών i. Αντίθετα, αν igp[i] == 0, η πληροφορία δεν συσχετίζεται με την ομάδα i.
- iprev και inext: Δύο δείκτες που δείχνουν στο επόμενο και στο προηγούμενο στοιχείο της λίστας.

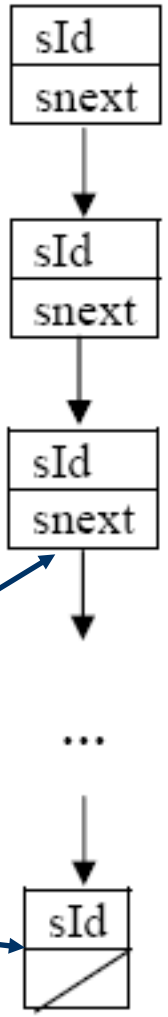



```
struct Info {
        int iId;
        int itm;
        int *igp;
        struct Info *iprev;
        struct Info *inext;
};
```



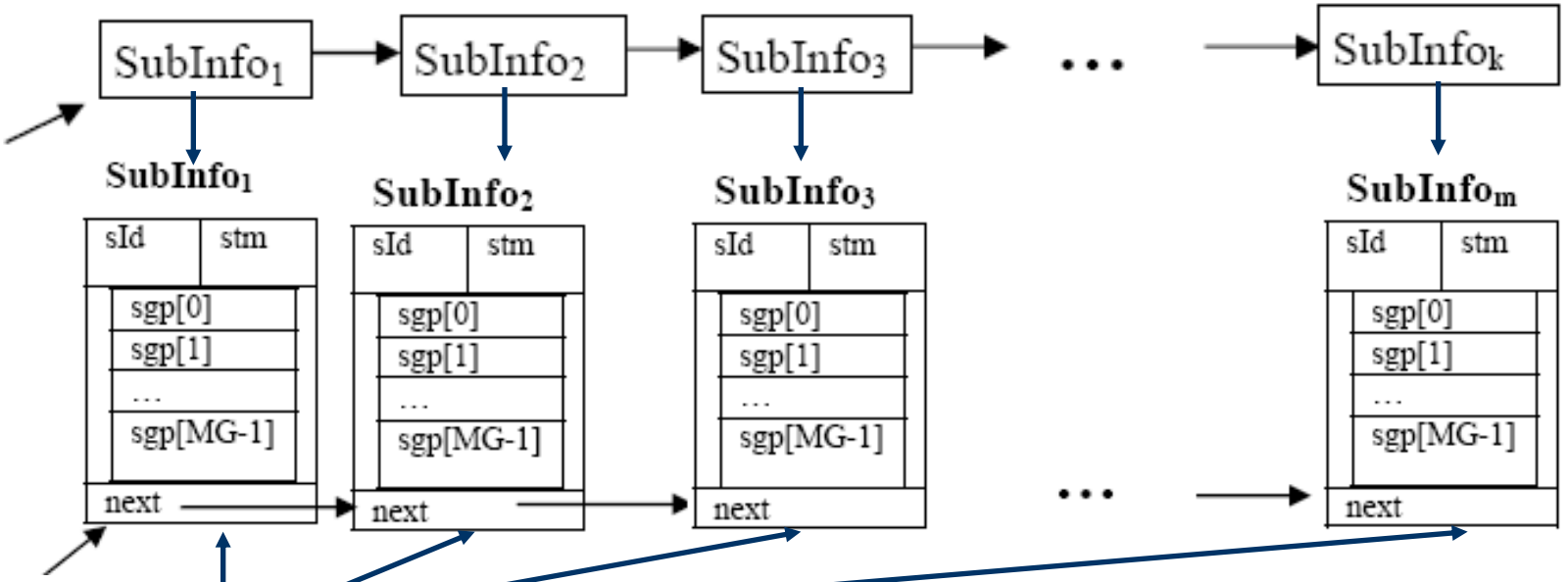
Groups:Λίστα Συνδρομητών Ομάδας

- sId: Ένας ακέραιος που αποτελεί το αναγνωριστικό του συνδρομητή που έκανε συνδρομή και
- Snext: Ένας δείκτης στο επόμενο στοιχείο της λίστας συνδρομών της ομάδας αυτής.




```
struct Subscription {
        int sId;
        struct Subscription *snext;
};
```

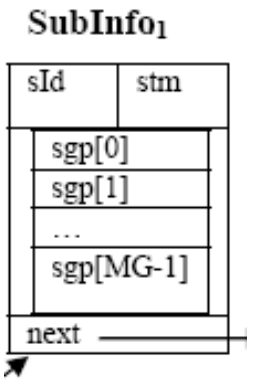
Συνδρομητές (1/2)




```
struct SubInfo {
        int sId;
        int stm;
        int *sgp;
                *snext;
};
```

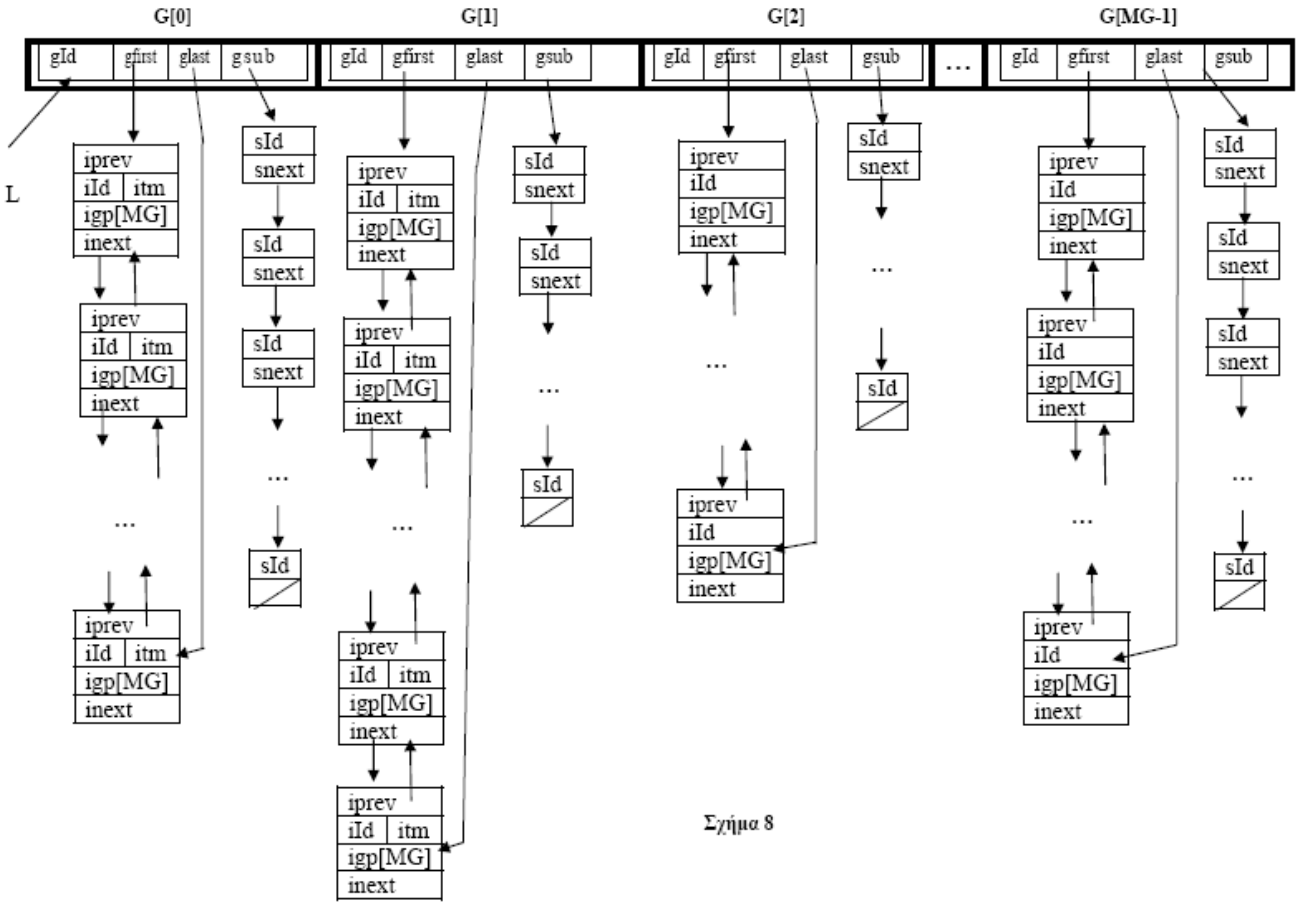
Συνδρομητές (2/2)

- έναν ακέραιο sId που αποτελεί το αναγνωριστικό του συνδρομητή.
- ο Έναν ακέραιο stm που αποθηκεύει την «χρονική στιγμή» στην οποία ο συνδρομητής καταφθάνει στο σύστημα Η λίστα συνδρομητών είναι ταξινομημένη* βάσει του πεδίου stm των εγγραφών της.
- Έναν πίνακα sgp[MG] ο οποίος περιέχει MG δείκτες, έναν για κάθε ομάδα. Έστω οποιαδήποτε ομάδα k, 0 ≤ k ≤ MG-1. Αν ο δείκτης sgp[k] μιας εγγραφής της λίστας συνδρομητών είναι NULL*, ο συνδρομητής δεν είναι εγγεγραμμένος στην ομάδα με αναγνωριστικό k. Διαφορετικά, ο δείκτης sgp[k] δείχνει σε ένα από τα στοιχεία της λίστας πληροφοριών της ομάδας G[k]*.
- Έναν δείκτη next στο επόμενο στοιχείο της λίστας συνδρομητών.



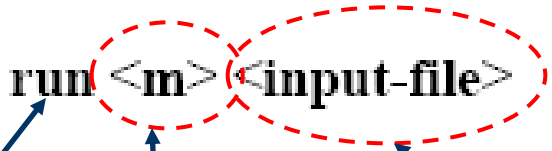

```
struct SubInfo {
        int sId;
        int stm;
        int *sgp;
                        *snext;
};
```

Συνολική «Αρχιτεκτονική»



Σχήμα 8

Εκτέλεση Προγράμματος



το όνομα του εκτελέσιμου αρχείου του προγράμματος

ένας ακέραιος που καθορίζει το μέγιστο πλήθος ομάδων (MG) στο σύστημα

είναι το όνομα ενός αρχείου εισόδου που περιέχει γεγονότα

Γεγονότα: Αφιξη πληροφορίας

- I <itm> <iId> <gId1> <gId2> … <gIdk> -1:
- Γεγονός τύπου Insert_Info το οποίο σηματοδοτεί την άφιξη νέας πληροφορίας
- <iTM>: Χρονική στιγμή εισόδου
- <iId>: Αναγνωριστικό πληροφορίας
- <gId1> <gId2> … <gIdk>: Οι ομάδες με τις οποίες συσχετίζεται η πληροφορία
- -1 : Τερματισμός λίστας ομάδων
- Οι λίστες πληροφοριών των ομάδων με τις οποίες συσχετίζεται η πληροφορία <iId> πρέπει να ενημερωθούν (προσοχή στην ταξινόμηση)

Γεγονότα: Εκτύπωση πληροφορίας

- <iTM> <iId> DONE
  - GROUPID = <gId1>, INFOLIST = <iId11, iId21, …, iIdr11>
  - GROUPID = <gId2>, INFOLIST = <iId12, iId22, …, iIdr22>
  - ...
  - GROUPID = <gIdk>, INFOLIST = <iId1k, iId2k, …, iIdrkk>

Γεγονότα: Αφιξη Συνδρομητή

S <sTM> <sId> <gId1> <gId2> … <gIdm> -1

- Γεγονός τύπου Subscriber_Registration το οποίο σηματοδοτεί την εγγραφή ενός νέου συνδρομητή
- <σTM>: Χρονική στιγμή εισόδου
- <sId>: Αναγνωριστικό συνδρομητή
- <gId1> <gId2> … <gIdk>: Οι ομάδες στις οποίες εγγράγεται ο συνδρομητής (προσοχή ανανέωση και της λίστας συνδρομητών και της λίστας συνδρομητών της ομάδας!!!)
- -1 : Τερματισμός λίστας ομάδων

Γεγονότα: Εκτύπωση Συνδρομητών

- S <sTM> <sId> <gId1> <gId2> … <gIdm> DONE
  - SUBSCRIBERLIST = <sId1, sId2, …, sIdm>
  - GROUPID = <gId1>, SUBLIST = <sId11, sId21, …, sIdn11>
  - GROUPID = <gId2>, SUBLIST = <sId12, sId22, …, sIdn22>
  - …
  - GROUPID = <gIdm>, SUBLIST = <sId1m, sId2m, …, sIdnmm>

Γεγονότα: Κατανάλωση πληροφοριών

- C <sId>:
  - <sId>: Το αναγνωριστικό του συνδρομητή
- Για κάθε ομάδα G[k] για την οποία ενδιαφέρεται ο συνδρομητής <sId>, θα πρέπει:
  - να διαβασθούν οι πιο πρόσφατες δημοσιευμένες πληροφορίες (μετά την τελευταία εκτέλεση της λειτουργίας κατανάλωσης του <sId> για το G[k]) μέσω του δείκτη sgp[k] που διατηρεί ο <sId>.
  - να εκτυπώνονται τα αναγνωριστικά των πληροφοριών που καταναλώνονται
  - να ενημερωθεί ο δείκτης sgp[k] ώστε να δείχνει στην πιο πρόσφατη πληροφορία της ομάδας G[k].

Γεγονότα: Εκτύπωση μετά από γεγονός κατανάλωσης

- C <sId> DONE
  - GROUPID = <gId1>, INFOLIST = <iId11, iId21, …, iIdr11>, NEWSGP = <iId1>
  - GROUPID = <gId2>, INFOLIST = <iId12, iId22, …, iIdr22>, NEWSGP = <iId2>
  - …
  - GROUPID = <gIdk>, INFOLIST = <iId1k, iId2k, …, iIdrkk>, NEWSGP = <iIdk>

Γεγονότα: Αποχώρηση Συνδρομητή

- D <sId>
  - <sId>: Το αναγνωριστικό του συνδρομητή
- Η εγγραφή που αναφέρεται στον συνδρομητή με αναγνωριστικό <sId> θα πρέπει:
  - να διαγραφεί από τη λίστα συνδρομητών.
  - να διαγραφεί η εγγραφή που αναφέρεται στη συνδρομή του εν λόγω συνδρομητή από κάθε ομάδα στην οποία έχει εγγραφεί ο συνδρομητής.

Γεγονότα: Εκτύπωση μετά από αναχώρηση συνδρομητή

- D <sId> DONE
  - SUBSCRIBERLIST = <sId1, sId2, …, sIdm>
  - GROUPID = <gId1>, SUBLIST = <sId11, sId21, …, sIdn11>
  - GROUPID = <gId2>, SUBLIST = <sId12, sId22, …, sIdn22>
  - …
  - GROUPID = <gIdk>, SUBLIST = <sId1k, sId2k, …, sIdnkk>

Γεγονότα: Εκτύπωση όλων των δομών

- P
- P DONE
  - GROUPID = <gId1>, INFOLIST = <iId11, iId21, …, iIdr11>, SUBLIST = < iId11, iId21, …, iIdn11>
  - GROUPID = <gId2>, INFOLIST = <iId12, iId22, …, iIdr22>, SUBLIST = < iId12, iId22, …, iIdn22>
  - …
  - GROUPID = <gIdk>, INFOLIST = <iId1k, iId2k, …, iIdrkk>, SUBLIST = < iId1k, iId2k, …, iIdnkk>
  - SUBSCRIBERLIST = <sId1, sId2, …, sIdm>
  - SUBSCRIBERID = <sId1>, GROUPLIST = <gId11, …, gIdv11>
  - SUBSCRIBERID = <sId2>, GROUPLIST = <gId12, …, gIdv22>
  - …
  - SUBSCRIBERID = <sIdm>, GROUPLIST = <gId1m, …, gIdvmm>
  - NO_GROUPS = <number of groups>, NO_SUBSCRIBERS = <number of subscribers>

Παραλλαγές Υλοποίησης: *1η Παραλλαγή*

- *1η Παραλλαγή*
- Το πρόγραμμα που θα δημιουργηθεί θα πρέπει να εκτελείται καλώντας την ακόλουθη εντολή: **run <m> <input-file>**
- Επειδή το πλήθος των ομάδων δεν είναι γνωστό εκ των προτέρων πριν την εκτέλεση του προγράμματος:
  - δεν μπορούμε πλέον να δεσμεύσουμε με στατικό τρόπο τον απαραίτητο χώρο μνήμης για την αποθήκευση των πινάκων που χρησιμοποιούν το MG (G, igp, sgp).
  - είναι απαραίτητο να δεσμεύεται χώρο μνήμης για τους πίνακες αυτούς κατά την εκτέλεση του προγράμματος, δηλαδή με δυναμικό τρόπο (κατάλληλη χρήση της συνάρτησης malloc).
- Η παραλλαγή αυτή επηρεάζει και τη μορφή των δομών δεδομένων που χρησιμοποιούνται στην εργασία. Η νέα μορφή των δομών αυτών παρουσιάζεται στη συνέχεια.

Παραλλαγές Υλοποίησης: $2^{η}$ *Παραλλαγή*

- θα πρέπει να δηλώνετε πως ο συνδρομητής δεν είναι εγγεγραμμένος στην ομάδα με αναγνωριστικό k, θέτοντας στο δείκτης sgp[k] την τιμή NULL.
  - η τιμή NULL του δείκτη sgp[k] μπορεί να σημαίνει:
    - ότι ο αντίστοιχος συνδρομητής δεν ενδιαφέρεται για τις πληροφορίες που δημοσιεύονται στην ομάδα k, είτε πως
    - ο συνδρομητής ενδιαφέρεται όμως η ομάδα k δεν περιέχει δημοσιευμένες πληροφορίες.
  - Η υλοποίηση του γεγονότος C δε μπορεί να γνωρίζει τη σημασία της τιμής NULL του sgp[k].
    - Έστω οποιαδήποτε ομάδα k, $0 \leq k \leq MG-1$, και <sId1k, sId2k, …, sIdmk> η λίστα συνδρομών της ομάδας αυτής, όπου sIdik, $1 \leq i \leq m$, οι συνδρομητές που ενδιαφέρονται για τις πληροφορίες της ομάδας k.
    - Όταν εισάγεται η πρώτη πληροφορία στην αρχικά κενή (από πληροφορίες) ομάδα k θα πρέπει :
      - να γίνεται διάσχιση της λίστας συνδρομών της ομάδας αυτής και να ενημερώνονται οι δείκτες sgpi[k], $1 \leq i \leq m$, της δομής δεδομένων κάθε συνδρομητή, ώστε να δείχνουν στην πρώτη πληροφορία της ομάδας k που μόλις εισήχθη.

HY240

Ενότητα 2η: Περιγραφή της main() και των input files

Main() in C (1/6)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LINE_LENGTH 64
#define MG 10

int main(int argc, char** argv)
{
    FILE *fp = NULL;
    char *token,*search=" ";
    char line[LINE_LENGTH], event;
    int itm,iId,stm,sId;
    int gId;
```

```
/* Check command line arguments */
    if( argc < 2 )
    {
        printf("\n Usage: %s <input-
    file>\n", argv[0]);
        return(EXIT_FAILURE);
    }
```

Main() in C (2/6)

```
/* Open input file */
    if( ( fp = fopen(argv[1], "r") ) == NULL )
    {
        printf("\n Could not open file: %s\n", argv[1]);
        return(EXIT_FAILURE);
    }

    /* Read input file line-by-line and handle the events */
    while( fgets(line, LINE_LENGTH, fp) != NULL )
    {
        /* Uncomment the line below when in debug mode */
        //printf("\n Event: %s ", line);
        switch(line[0])
        {
```

Main() in C (3/6)

```
/* Insert_Info event */
      case 'I':
                sscanf(line, "%c %d %d", &event, &itm, &iId);
                printf("\nEvent : %c itm : %d iId : %d ",event,itm,iId);
                token=strtok(line,search);
                token=strtok(NULL,search);
                token=strtok(NULL,search);
                while ((gId=atoi(token=strtok(NULL,search))) != -1){
                printf ("gId : %d ",gId);
                /* your code... */
                }
                /* your code... */
                break;
```

Main() in C (4/6)

```
/* Subscriber_Registration event */
      case 'S':
            sscanf(line, "%c %d %d", &event, &stm, &sId);
            printf("\nEvent : %c stm : %d sId : %d ",event,stm,sId);
            token=strtok(line,search);
            token=strtok(NULL,search);
            token=strtok(NULL,search);
            while ((gId=atoi(token=strtok(NULL,search))) != -1){
                  printf ("gId : %d ",gId);
                  /* your code... */
            }
            /* your code... */
            break;
```

Main() in C (5/6)

```
/* Consume event */
      case 'C':
            sscanf(line, "%c %d", &event, &sId);
            printf("\nEvent : %c sId : %d",event,sId);
            /* your code... */
      break;

      /* Delete_Subsciber event */
      case 'D':
            sscanf(line, "%c %d", &event, &sId);
            printf("\nEvent : %c sId : %d",event,sId);
            /* your code... */
      break;
```

Main() in C (6/6)

```
/* Print event */
      case 'P':
                sscanf(line, "%c", &event);
                printf("\nEvent : %c ",event);
                /* your code... */
      break;
      /* Ignore everything else */
      default:
                /* Uncomment the line below when in debug mode */
                /* printf("\n Ignoring line: %s", line); */
                break;
                }
      }
      printf("\n");
      return (EXIT_SUCCESS);
```

Main() in Java (1/6)

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Main {

    public static void main(String[] args) {

        BufferedReader file_input = null;
        String line = new String();
        char event;
        int i, itm, iId, stm, sId, gId;

        // Check command line arguments
        if (args.length < 1) {
            System.out.println("\n Usage: <program_name> <input-file>\n");
            System.exit(0);
        }
```

Main() in Java (2/6)

```
try {
        // Open input file
        file_input = new BufferedReader(new FileReader( args[1] ));

        if( file_input == null ) {
            System.out.println("\n Could not open file:" + args[1] + "\n");
            System.exit(1);
        }// Read input file line-by-line and handle the events

        int numRead = 0;
        String[] params = null;
    while ((line = file_input.readLine()) != null) {
            if(line.equals("")) continue;
            // Uncomment the line below when in debug mode
            //System.out.println("\n Event: \"" + line + "\"");
```

Main() in Java (3/6)

```
switch(line.charAt(0)) {
                    // Insert_Info event
                case 'I':
                    params = line.split(" ");
                    event = params[0].charAt(0);
                    itm = Integer.parseInt(params[1]);
                    iId = Integer.parseInt(params[2]);
                    for (i=3; (gId = Integer.parseInt(params[i]))!=-1; i++) {
                        // your code...
                        ;
                    }
                    // your code...
                    break;
```

Main() in Java (4/6)

```
// Subscriber_Registration event
                case 'S':
                    params = line.split(" ");
                    event = params[0].charAt(0);
                    stm = Integer.parseInt(params[1]);
                    sId = Integer.parseInt(params[2]);
                    for (i=3; (gId = Integer.parseInt(params[i]))!=-1; i++) {
                        // your code...
                        ;
                    }
                    // your code...
                    break;
```

Main() in Java (5/6)

```
// Consume event
 case 'C':
     params = line.split(" ");
     event = params[0].charAt(0);
     sId = Integer.parseInt(params[1]);
      // your code...
     break;
  // Delete_Subsciber event
 case 'D':
     params = line.split(" ");
     event = params[0].charAt(0);
     sId = Integer.parseInt(params[1]);
    // your code...
     break;
```

Main() in Java (6/6)

```
// Print event
                case 'P':
                   params = line.split(" ");
                   event = params[0].charAt(0);
                    // your code...
                   break;
                 default: // Ignore everything else
                   // Uncomment the line below when in debug mode
                   //System.out.println("\n Ignoring line:\"" + line + " \"");
                   break;
            }
        }
        file_input.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
  }
```

Test file (1/2)

I 0 0 0 1 5 -1
I 1 1 1 2 3 -1
S 2 0 5 -1
S 3 1 0 1 2 3 4 -1
I 4 2 4 -1
I 5 3 0 2 4 -1
S 6 2 0 2 3 -1
D 0
C 1
P
//Infos : gId0=2, gId1=2, gId2=2, gId3=1, gId4=2, gId5=1
//Subscriptions : gId0=2, gId1=1, gId2=2, gId3=2, gId4=1, gId5=0
//Subscribers : sId1=<gId0,gId1,gId2,gId3,gId4>, sId2=<gId0,gId2,gId3>

Test file (2/2)

S 7 3 1 3 -1
C 2
I 8 4 1 3 -1
D 1
S 9 4 1 -1
I 10 5 1 2 4 5 -1
C 2
C 4
D 3
P
//Infos : gId0=2, gId1=4, gId2=3, gId3=2, gId4=3, gId5=2
//Subscriptions : gId0=1, gId1=1, gId2=1, gId3=1, gId4=1, gId5=0
//Subscribers : sId2=<gId0,gId2,gId3>, sId4=<gId1>

HY240

Ενότητα 3η: Δυναμική Υλοποίηση στοίβας

Στοίβα#1 (Ορισμοί)

```
#include<stdio.h>
#include<malloc.h>
#define maxsize 10

void push();
void pop();
void top();
void display();

struct node
{
      int info;
      struct node *link;
}*start=NULL, *new,*temp,*p;

typedef struct node N;
```

Στοίβα #1 (Main)

```
main(){
	int ch,a;
	while(1){
		printf("\t\t\tEpile3te leitourgia");
		printf("\n 1.Push");
		printf("\n 2.Pop");
		printf("\n 3.Top");
		printf("\n 4.Display");
		printf("\n 5.Exit");
		printf("\n Enter your choice : ");
		scanf("%d",&ch);
		switch(ch){
			case 1: push(); break;
			case 2: pop(); break;
				case 3: top(); break;
			case 4: display(); break;
			case 5: exit(0);
			default: printf("\nInvalid choice"); break;
		}//end switch
	}//end while
}//end main
```

Στοίβα #1 (Push)

```
void push()
{
	new=(N*)malloc(sizeof(N));
	printf("\nEnter the item : ");
	scanf("%d",&new->info);
	new->link=NULL;
	if(start==NULL)
	{
		start=new;
	}
	else
	{
		p=start;
		while(p->link!=NULL)
			p=p->link;
		p->link=new;
	}
}
```

Στοίβα #1 (Pop)

```
void pop()
{
	if(start==NULL)	{
		printf("\nStack is empty");
	}
	else if(start->link==NULL){
		printf("\nThe deleted element is : %d",start->info);
		free(start);
		start=NULL;
	}
	else{
		p=start;
		while(p->link!=NULL){
			temp=p;
			p=p->link;
		}
		printf("\nDeleted element is : %d\n", p->info);
		temp->link=NULL;
		free(p);
	}
}
```

Στοίβα #1 (Top)

```
void top()
{
	if(start==NULL)
	{
		printf("\nStack is empty");
	}
	else if(start->link==NULL)
	{
		printf("\nThe Top element is : %d\n",start->info);
	}
	else
	{
		p=start;
		while(p->link!=NULL)
		{
			p=p->link;
		}
		printf("\nThe top element is : %d\n", p->info);
	}
}
```

Στοίβα #1 (Display)

```
void display()
{
	if(start==NULL)
		printf("\nStack is empty");
	else
	{

		printf("\nThe elements are : ");
		p=start;
		while(p!=NULL)
		{
			printf("%d ",p->info);
			p=p->link;
		}
		printf("\n");
	}
}
```

Στοίβα#2 (Ορισμοί)

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

#define maxsize 10

struct node
{
	int info;
	struct node *link;
}*start=NULL;
typedef struct node N;

N* push(N*);
N* pop(N*);
void top(N*);
void display(N*);
```

Στοίβα #2 (Main)

```
main(){
      N* p;  int ch,a;
      while(1){
                  printf("\t\t\tEpile3te leitourgia");
                  printf("\n 1.Push");
                  printf("\n 2.Pop");
                  printf("\n 3.Top");
                  printf("\n 4.Display");
                  printf("\n 5.Exit");
                  printf("\n Enter your choice : ");
                  scanf("%d",&ch);
                  switch(ch){
                  case 1:  p = push(p); break;
                  case 2:  p = pop(p); break;
                  case 3: top(p); break;
                  case 4: display(p); break;
                  case 5: exit(0);
                  default: printf("\nInvalid choice"); break;
                  }
      }
}
```

Στοίβα #2 (Push)

```
N* push(N* p){
	N *new=(N*)malloc(sizeof(N));
	printf("\nEnter the item : ");
	scanf("%d",&new->info);
	new->link=NULL;
	if(start==NULL)
	{
		start=new;
	}
	else
	{
		p=start;
		while(p->link!=NULL)
			p=p->link;
		p->link=new;
	}

	return p;
}
```

Στοίβα #2 (Pop)

```
N* pop(N* p){
   N * temp;
      if(start==NULL){
                  printf("\nStack is empty");
      }
      else if(start->link==NULL){
                  printf("\nThe deleted element is : %d",start->info);
                  free(start);
                  start=NULL;
      }
      else{
                  p=start;
                  while(p->link!=NULL)
                  {
                              temp=p;
                              p=p->link;
                  }
                  printf("\nDeleted element is : %d\n", p->info);
                  temp->link=NULL;
                  free(p);
      }
      return p;
}
```

Στοίβα #2 (Top)

```
void top(N* p){
	if(start==NULL){
		printf("\nStack is empty");
	}
	else if(start->link==NULL){
		printf("\nThe Top element is : %d\n",start->info);
	}
	else{
		p=start;
		while(p->link!=NULL)
		{
			p=p->link;
		}
		printf("\nThe top element is : %d\n", p->info);
	}
}
```

Στοίβα #2 (Display)

```
void display(N* p){
    if(start==NULL)
        printf("\nStack is empty");
    else{

        printf("\nThe elements are : ");
        p=start;
        while(p!=NULL)
        {
            printf("%d ",p->info);
            p=p->link;
        }
        printf("\n");
    }
}
```