CS240: Data Structures
Fall Semester – Academic Year 2017-2018
Professor: Panagiota Faturu
Project - 1st Phase

Due Date: Monday, 20 November 2017, time 23:59.

Turnin Method: Using the turnin program. Information about how turnin works are supplied by the course website.

General Description

In this project you are asked to create a program that simulates the Trojan War, between Greeks (Achaeans by Homer) and Trojans, under the walls of Troy. The events of this project concern the concentration of the Greeks for campaigning and redeeming the city thanks to the trick of the Trojan Horse. This war is one of the main events of Greek Mythology and was the source of inexhaustible inspiration for ancient Greek literature, including the works of Homer: the Iliad and the Odyssey.

Detailed Description of Requested Implementation

The list of soldiers (Rhapsody B, Day 22). "*During the preparations, the poet invokes the help of the Muses to quote the long list of the warriors*" (Homeric Epics: Iliad II Secondary High School). One of the tasks of this project is the implementation of the warriors' registration. Information about the warriors of the Achaean camp is stored in a singly linked list, unsorted, with a sentinel node. This list is called **registration list**. Each node in this list is a record of type soldier with the following fields:

- **sid:** Identity (of type int) that uniquely identifies the soldier.
- **gid:** Identity (of type int) that uniquely identifies the general to which the soldier obeys.
- **next:** Pointer (of type soldier) to the next node of the registration list.

Figure 1 shows the registration list.

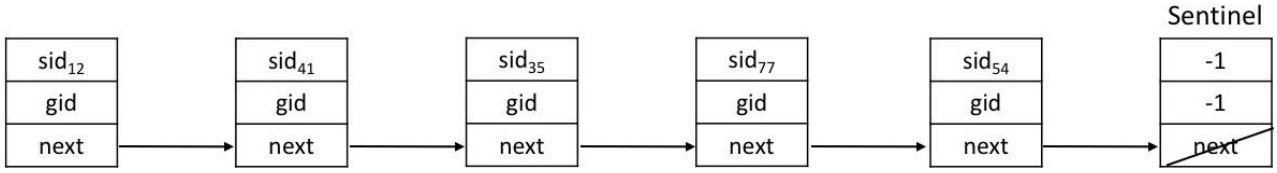

Figure 1: The singly linked registration list, which is unsorted, with a sentinel node.

The troops of the Achaeans are organized in groups of leading brave kings and generals (such as Agamemnon, Menelaus, Nestor, Odysseus, Achilles, Diomedes and others). Each king / general has his own army (a group of soldiers) camped by the ships that carried the soldiers to Troy. Information about kings / generals is stored in a singly linked, unsorted list, with a sentinel node. This list is called **generals list**. Each node in this list is a record of type general with the following fields:

- **gid:** Identifier (of type int) that uniquely identifies the general.
- **combats_no:** Number (of type int) representing the battles in which the general has participated.
- **soldiers_head:** Pointer (of type DDL_soldier) in the first node of the list of soldiers who obey the general. This list is double linked, sorted by the soldiers' identifier and is called **soldiers list**. Each node in this list is a record of type DDL_soldier as described above.
    - **sid**: Identifier (of type int) that uniquely characterizes the soldier.
    - **gid**: Identifier (of type int) that uniquely characterizes the king / general to whom the soldier obeys.
    - **next**: Pointer (of type soldier) at the next node in the registration list.
    - **prev**: Pointer (of type soldier) at the previous node in the registration list.
- **soldiers_tail:** Pointer (of type DDL_soldier) at the last node of the **list of soldiers** who obey the general.
- **next:** Pointer (of type general) to the next node in the list of generals.

The sentinel node is of type DDL_soldier with a value of -1 in the gid and combats_no fields, while the soldiers_head and soldiers_tail fields have a NULL value.

Battles in Rhapsodies D and E (22nd day), Theta (25th day), L, M and N (26th day, battles around the Achaean wall), P and R (27th day, Patroklos leads the Myrmedons to battle), F (Achilles returns to the battle and faces, apart from the Trojans, the river Skamandros that rushes against him).

Another of the tasks of this work is to implement the battles. Each of the battles involves some of their generals and soldiers. Soldiers battling are stored in a combat structure containing the following fields:

- **soldiers_cnt**: Integer which represents the number of soldiers involved in the battle.
- **combat_s**: Pointer (of type c_soldier) at the beginning of a singly linked, unsorted **battle list**. Each item in the list is a c_soldier record with the following fields:
    - **sid**: Identifier (of type int) which uniquely characterizes the soldier.
    - **alive**: Number (of type int) which represents the status of the soldier. The value 1 indicates that the soldier is alive, while the value 0 is the opposite.
    - **gid**: Identifier (of type int) which uniquely characterizes the general to whom the soldier obeys.
    - **next**: Pointer (of type c_soldier) to the next node on the battle list.

Figure 2 shows the list of generals, and the list of soldiers pointed to by each item in the list of generals.

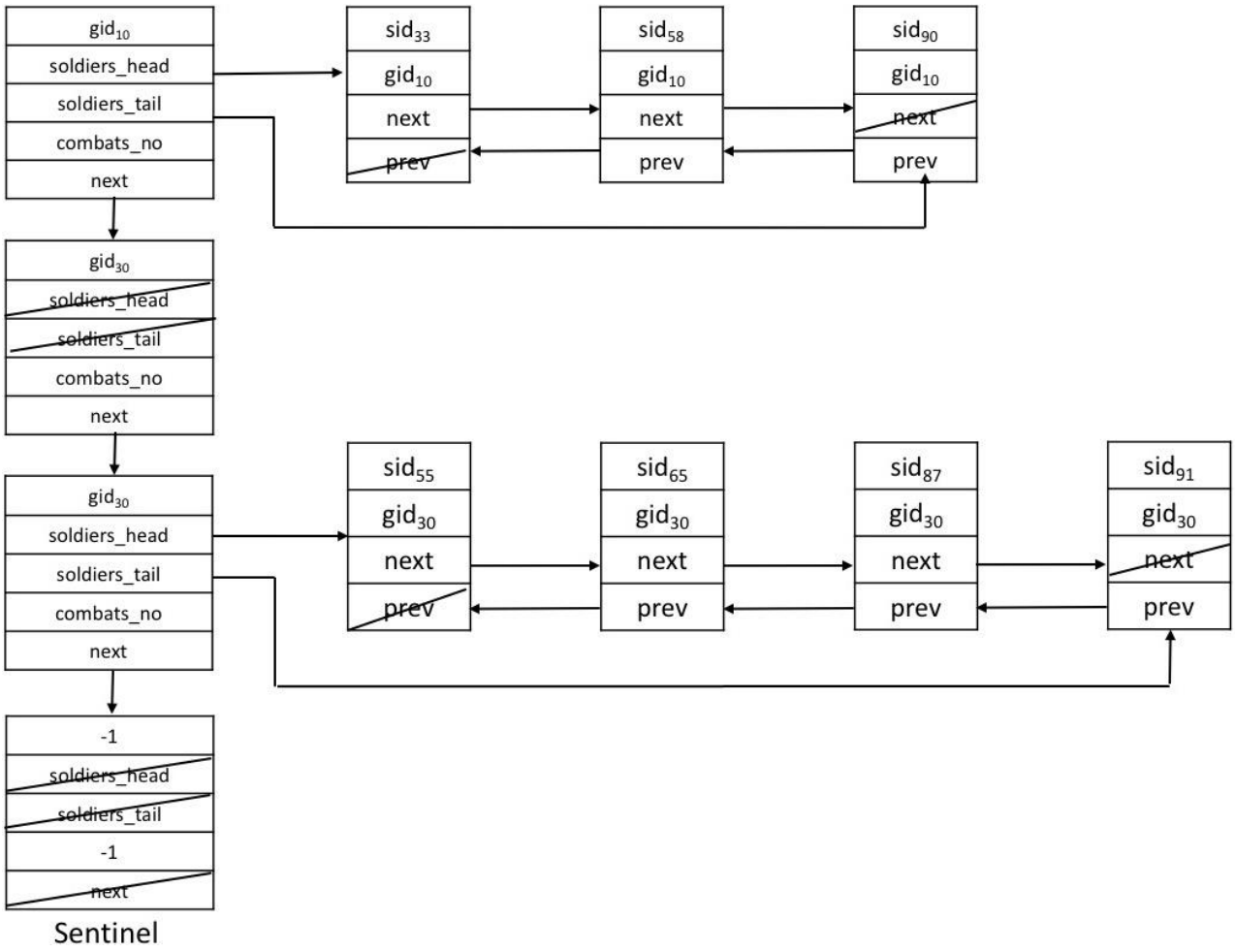

Figure 2: The unsorted, singly linked list of generals. Each node (general) contains a double-linked soldier list. The list of soldiers of each general is sorted based on the soldier's identifier.

Figure 3 shows the combat structure containing the battle list and the counter of soldiers participating in the battle.

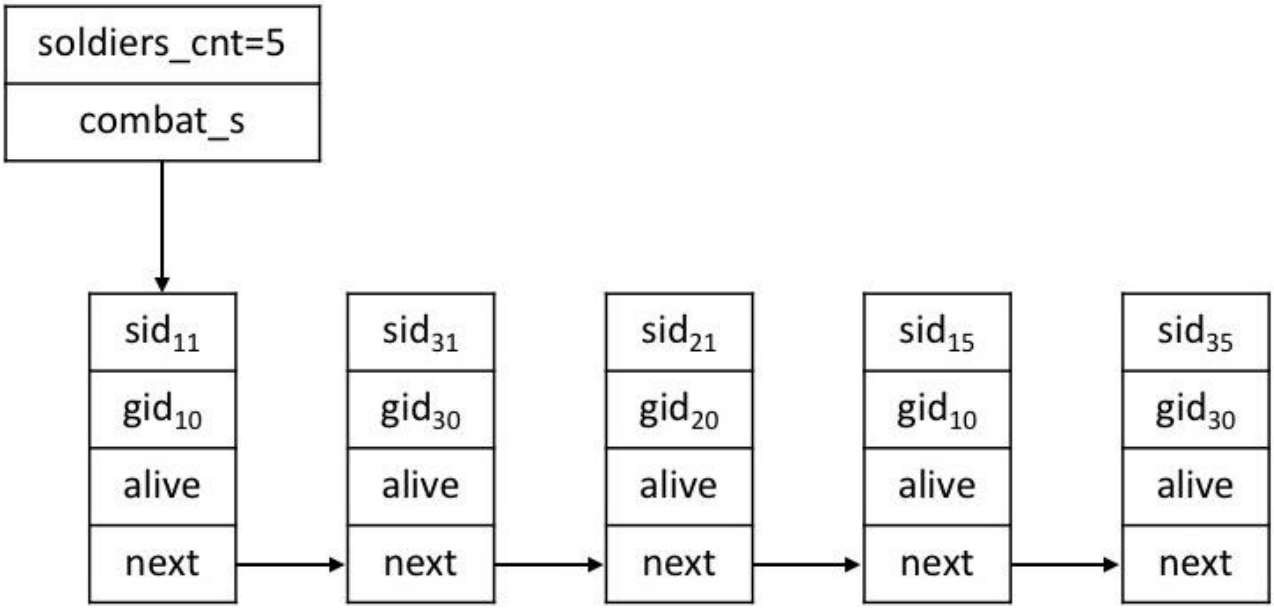

Figure 3: The combat structure containing the battle list and the counter of soldiers participating in the battle

How the Program Works

The program to be created should be executed by calling the following command:

<executable> <input-file>

where <executable> is the name of the executable file of the program (eg a.out) and <input-file> is the name of an input file (eg testfile) containing events of the following formats:

- **R <sid> <gid>**

Event indicating that a soldier is inscribed on the battlefield **(Rhapsody B [48-877], Day 22)**. This inserts a new soldier to the registration list. The soldier will have the <sid> identifier and will act under the command of the general with a <gid> identifier. Upon completion of such an event, the program should print the following information:

```
R <sid> <gid>
    Registration list = <sid1:gid1>, <sid2:gid2>, ..., <sidn:gidn>
DONE
```

where n is the number of nodes in the registration list and for each $i \in \{1, \ldots, n\}$, $<sid_i>$ is the soldier's identifier corresponding to the i-th node in that list and $<gid_i>$ the identifier of the general who the i-th node soldier obeys.

- **G <gid>**

Event indicating that the king/general with identifier <gid> participates in the campaign. This inserts a node of type general into the list of generals. At first the soldier list of the general will be empty. The insertion must be done with the least time complexity, given that the <gid> that follows each G event in the test files is unique. Upon completion of such an event, the program should print the following information:

```
G <gid>
    Generals = <gid1>, <gid2>, ..., <gidn>
DONE
```

where n is the number of nodes in the list of generals and for each $i \in \{1, ..., n\}$, $<gid_i>$ is the general id corresponding to the i-th node in that list.

- **D**

Distribute soldiers type event that marks evening rest. The soldiers reside in the camps they belong to (according to the general they obey). By doing so, you must traverse the registration list and create the lists of soldiers belonging to each general. This can be done by looking at the <gid> field of each registration list node and then search for that general id in the list of generals. Then we will insert a struct type DDL_soldier which will correspond to the soldier in the list of soldiers of the general. For example, if the <gid> field of a registration list node has a value 32, then the general with id 32 will be searched in the list of generals. A new

struct DDL_soldier will then be created for the soldier in the general list of soldiers we previously identified. Additionally, after each insertion, the list of soldiers of each general must be sorted based on the soldiers' identifier. At the end of the event, for all soldiers in the registration list, there must be corresponding entries in the appropriate lists of generals' soldiers. **This process must be executed at time O(n), where n is the number of soldiers contained in the registration list.**

Upon completion of such an event, the program should print the following information:

```
D
    GENERALS:
    <gid1>: <sid1,1> . . . <sid1,n1>
    <gid2>: <sid2,1> . . . <sid2,n2>
    ...
    <gidk>: <sidk,1> . . . <sidk,nk>
DONE
```

where for each i, $1 \leq i \leq k$, n_i is the size of the list of soldiers in the i-th node of the list of generals, and for each j, $1 \leq j \leq n_i$, και <$sid_{i,j}$> is the identifier of the j-th node in the list of soldiers of the i-th general.

- **M <gid_1> <gid_2>**

Event that marks the departure of Achilles from the battle (after his quarrel with Agamemnon), while Patroclos convinces Achilles to lead the Myrmoneses into the battle.

Consider that Achilles has an identifier <gid_1> and Patroklos has an identifier <gid_2>. You must delete the node with identifier <gid_1> from the list of generals. In addition, Achilles' soldiers will be transferred to Patrocklus soldiers' list (who has the identifier <gid_2>[1]) **The merging of the two lists must be done with complexity O(n_1 + n_2), where n_1 and n_2 the size of the list of soldiers of the first and second generals, respectively**. In other words, you must traverse each of the two lists of soldiers only once. After the end of the process, the list of soldiers of the general with identifier <gid2> must be sorted.

After the completion of such an event, the program should print the following information:

```
M
    GENERALS:
    <gid1>: <sid1,1> . . . <sid1,n1>
    <gid2>: <sid2,1> . . . <sid2,n2>
    ...
    <gidk>: <sidk,1> . . . <sidk,nk>
DONE
```

1 *In the Iliad, Patroklos leads the Myrmondons into the battle by wearing Achilles' archery. However, in order to serve the educational objectives of the course, consider that Patroklos had his own men who participated in the battle along with the Myrmoneses.*

where for each i, $1 \leq i \leq k$, n_i is the size of the list of soldiers in the i-th node of the list of generals, and for each j, $1 \leq j \leq n_i$, και $<sid_{i,j}>$ is the identifier of the j-th node in the list of soldiers of the i-th general.

- **P <gid$_1$> <gid$_2$> <gid$_3$>**

Prepare for battle type event which marks the selection of battalions to participate in the next battle. The battle will include the generals with identifiers <gid1> <gid2> and <gid3> with their respective soldiers. Also for each of the generals <gid1>, <gid2> and <gid3>, you will increase the combat counter combats_no, indicating the number of battles that each general has participated in.

The soldiers who are going to fight will have to be stored in the battle list of the combat record. Each item in this list contains a c_soldier record. Each time a node is inserted into this list, you must properly update the soldiers_cnt counter of the combat record. The way in which the merging will take place is by selecting soldiers from the lists of generals alternately.

Step 1: First you will choose the first soldier of the first general, then the second general's first soldier and then the first soldier of the third general.

Step 2: Then select soldiers from the end of the list of soldiers of each general alternately. So in the fourth position of the battle list we will insert the last soldier of the first general, then the last soldier of the second general and then the last soldier of the third general.

By completing the selection of soldiers from the end of the list of soldiers of each general, you will continue with the selection of soldiers from the beginning of each list as described in step 1. Each time you proceed from the point where you stopped the last time you performed the step 1. Next, you will perform step 2 again. This process will continue until you have added all soldiers from the list of each <gid1>, <gid2> and <gid3> in the battle list. In other words, you should run each list of generals <gid1>, <gid2> and <gid3> only once.

For each soldier you insert in the battle list of the combat structure, you set the value of alive = 1 to the soldier being alive.

Figure 4 shows an example of the lists of soldiers of generals with identifiers 10, 30 and 20 who will participate in the battle. Figure 5 shows the result of merging the soldiers of the above generals to the battle list according to the algorithm described above.

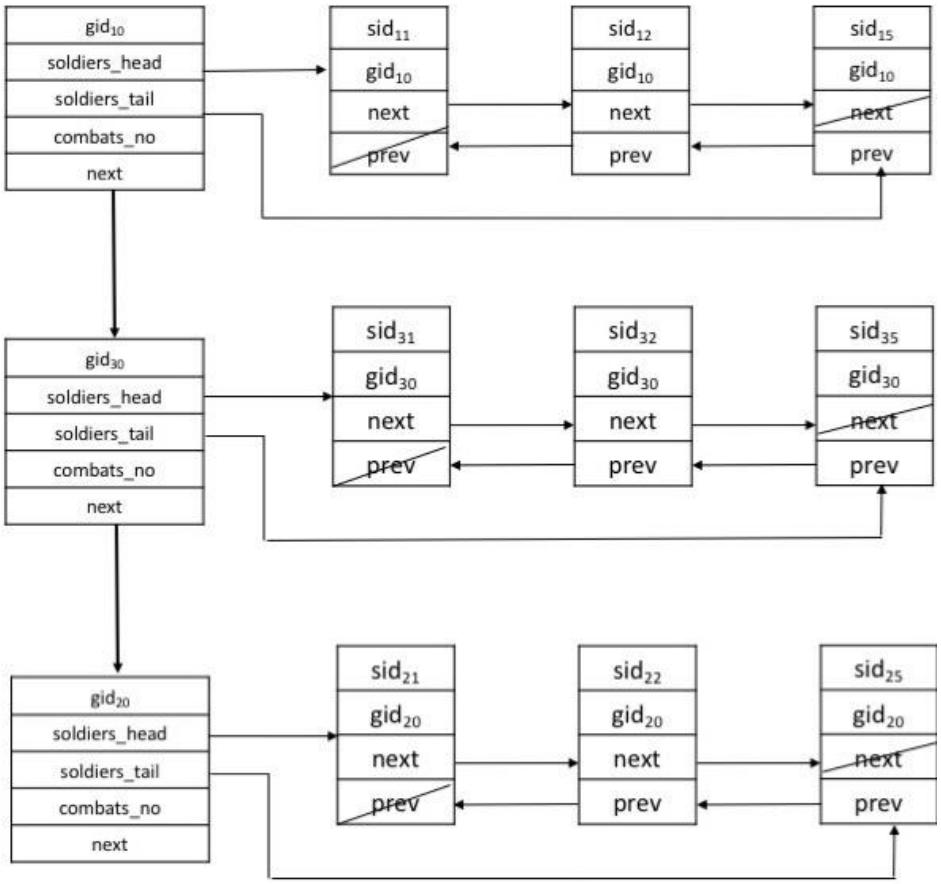

Figure 4 The list of generals containing generals with identifiers 10, 30 and 20 as well as the corresponding lists of soldiers they contain.

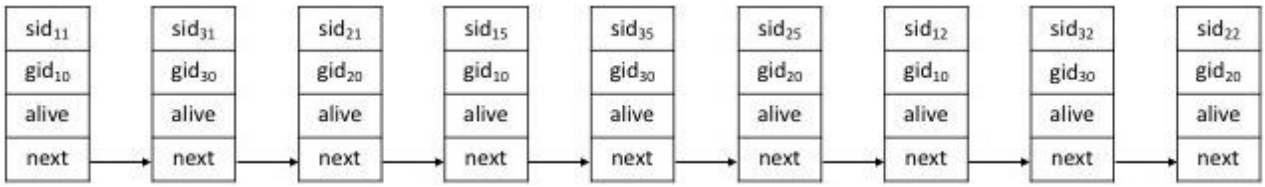

Figure 5 The battle list as it results from the merging of soldiers lists shown in Figure 4.

Upon completion of such an event, the program should print the following information:

```
P <gid1> <gid2> <gid3>
    Combat soldiers: <sid1>, <sid2> . . . <sidn>
DONE
```

where n is the size of the battle list and for each i, $1 \leq i \leq n$, $\langle sid_i \rangle$ is the i-th soldier's identifier on the battle list.

- **B <god_favor>**

Event that marks the battle simulation (see, for example, Phapsodies D, E, L, M, and N).

Battle simulation is signaled in this event (which is a of type battle). During the battle some of the fighters die. The size of losses is affected by the favor of the gods towards the Greeks as defined by the <god_favor> parameter.

If the gods do not favor Greek troops (eg in the battles of the 25th and 26th day, including the battles around the Achaean walls, Rhapsodies R, L, M, and N), then the losses amount to 40% of soldiers participating in the battle. Using the soldiers_cnt counter of the combat record, you will calculate the number of soldiers to be lost in the battle. Then you will traverse the combat_soldiers list of the combat record and set 40% of the first soldiers on the list as dead (you will set the alive field of combat_soldier value 0).

If Gods favor the Greek troops (eg battles in Phapsodies D, E, Y and F Rays), then the losses are less and correspond to 10% of the soldiers participating in the battle. In this case the way you choose which soldiers will lose their lives is as follows:

The soldiers involved in the battle are rowed. Each series consists of 10 soldiers. Losses are set at 10% of each row. In other words, you will run the battle_soldiers of the combat structure and for each 10th soldier you will mark the first soldier of the series dead.

Upon completion of such an event, the program should print the following information:

```
B <god_favor>
    Combat soldiers: <sid1:alive1>, <sid2:alive2>, . . . <sidn:aliven>
DONE
```

where n is the size of the battle list and for each i, $1 \leq i \leq n$, <sid_i> και <$alive_i$> is the identifier and status of the soldier corresponding to the i-th node in the battle list.

- **U**

Truce and burial of the dead (day 23, Rhapsody H381-432). Event of burial of the dead. In this event, the soldiers who have lost their lives in the battle should be deleted from the lists of soldiers of generals. In this case, you will traverse the battle list of the combat record and for every soldier who is dead (alive = 0 field), you will find out the general he belongs to (by looking at the c_soldier gid field) and then delete it from his general's list. In this process, you only have to traverse the battle list and the list of soldiers of each general. After the end of this process, the battle list of the combat record must be left empty and the soldiers_cnt counter must be equal to zero.

Additionally, you must traverse the registration list and delete the entry that corresponds to the soldier who died as well.

Upon completion of such an event, the program should print the following information:

```
U
    GENERALS:
    <gid1>: <sid1,1> . . . <sid1,n1>
    <gid2>: <sid2,1> . . . <sid2,n2>
    ...
    <gidk>: <sidk,1> . . . <sidk,nk>
DONE
```

where for each i, $1 \leq i \leq k$, n_i is the size of the list of soldiers in the i-th node of the list of generals, and for each j, $1 \leq j \leq n_i$, και $<sid_{i,j}>$ is the identifier of the j-th node in the list of soldiers of the i-th general.

– **T**

Trojan horse type event simulating the construction of the Trojan Horse. Five generals with most battles enter the Trojan Horse. To do this, you will use an array of size 5 to store pointers to the nodes of the list of generals.

In particular, you will do the following:

i. The array is initialized with the first 5 generals of the list of generals.
ii. As you traverse the list of generals, for each node you visit, you are looking at whether the combats counter has a value greater than the least-combat general in the array. If this happens, then we replace that general in the array with the current node in the list. Next, you have to re-calculate who is the least-combat general of the array and repeat step.
iii. After the end of the traversal of the list of generals, the array will contain pointers on the list nodes corresponding to the 5 generals with most battles.

Upon completion of such an event, the program should print the following information:

```
T
    General = <gid1:combats1>, <gid2:combats2>, ..., <gid5:combats5>,
DONE
```

and for each $i \in \{1, \ldots, 5\}$, $<gid_i>$ κα $<combats_i>$ is the identifier and the number of battles respectively of the general corresponding to the i-th position of the array.

– **X**

A *print generals* event that marks the printing of all generals. For each general, you must print all his details, including his list of soldiers. After the completion of such an event, the program should print the following information:

```
X
    GENERALS:
    <gid1>: <sid1,1> . . . <sid1,n1>
    <gid2>: <sid2,1> . . . <sid2,n2>
    ...
    <gidk>: <sidk,1> . . . <sidk,nk>
DONE
```

where for each i, $1 \leq i \leq k$, n_i is the size of the list of soldiers in the i-th node of the list of generals, and for each j, $1 \leq j \leq n_i$, και $<sid_{i,j}>$ is the identifier of the j-th node in the list of soldiers of the i-th general.

- **Y**

Event that marks the printing of all soldiers in the registration list. Upon completion of such an event, the program should print the following information:

```
Y
    Registration list = <sid1:gid1>, <sid2:gid2>, ..., <sidn:gidn>
DONE
```

where n is the number of nodes in the registration list and for each $i \in \{1, \ldots, n\}$, $<sid_i>$ is the soldier's identifier corresponding to the i-th node in that list and $<gid_i>$ the identifier of the general who the i-th soldier obeys to.

Data structures

In your implementation you are not allowed to use ready-made data structures provided by the language and/or libraries (eg, ArrayList in Java) whether the implementation is in C or Java. The C structures to be used for the implementation of this paper are presented below.

```
struct soldier {
       int sid;
       int gid;
       struct soldier *next;
};

struct DDL_soldier {
       int sid;
       int gid;
       struct soldier *next;
       struct soldier *prev;
};
```

```
struct general {
	int gid;
	int combats_no;
	struct DDL_soldier *soldiers_head;
	struct DDL_soldier *soldiers_tail;
	struct general *next;
};

struct c_soldier {
	int sid;
	int alive;
	int gid;
	struct c_soldier *next;
};


struct combat {
	int soldier_cnt;
	struct combat_soldier *combat_s;
};

/* global variable pointing to the beginning of the registration list */
struct soldier * registration_list;

/* global variable for the sentinel of registration list */
struct soldier *registration_sentinel;

/* global variable, pointer to the beginning of the generals list*/
struct general *generals_list;

/* global variable for the sentinel of generals list */
struct general *generals_sentinel;

/* global variable holding the combat info */
```