CS1100 - Lecture 28

Instructor : Jasine Babu

Teaching Assistants : Nikhila K N, Veena Prabhakaran

We will extend the program we developed at the end of last lecture to a program for finding the topper among a list of students. Recall that, we had defined a student structure using the following definition.

```
struct student
{  int roll;
   unsigned int marks[2];
   unsigned int total;
   char name[15];
};
```

Let us also define a constant `SIZE` as follows.

```
#define SIZE 150
```

An outline of a `main()` function for finding the topper can be written as follows.

```
int main()
{
   struct student l[SIZE];
   struct student s;
   int i, n;
   printf("enter number of students (<%d) \n", SIZE);
   scanf("%d",&n);
   if(n > SIZE)
   {
      printf("error \n");
      return 0;
   }
   for(i=0; i<n; i++)
   {
      printf("\nGive roll number, name, mark1, mark2 of student %d\n", i+1);
      l[i]=inputStudentDetails();
      printf("\n");
      updateTotal(&l[i]);
   }
   s=getTopper(l, n);
   printf("\nDetails of the topper \n");
   printf("\n Roll No.\tName\t\tMark1\tMark2\tTotal\n");
```

```
    printStudentDetails(s);
}
```

An array `l` of type `struct student` with enough space for storing the details of `150` students is declared in the `main()` function.

The `main()` function invokes four functions; `inputStudentDetails()`, `getTopper()`, `updateTotal()` and `printStudentDetails()`. As discussed in the previous lecture, the function `inputStudentDetails()` accepts `roll, name, marks[0], marks[1]` of a student from the terminal, stores it into a variable of type `struct student` and returns this variable. The instruction `l[i]=inputStudentDetails();` is intended to read the details of the i^{th} student and stores it into the location of `l[i]`. The function `updateTotal()` takes a pointer `ps` that points to a variable of type `struct student` as parameter and update the details of the student variable whose address is stored in `ps`. The function call `updateTotal(&l[i]);` is used to update total mark of the student whose details are stored in `l[i]`. The function `getTopper()` takes an array `sl` of type `struct student` and an integer `n` representing the number of students in the list and returns a variable of type `struct student` that stores the details of the student who stored the maximum total marks. After the function call `s=getTopper(l, n);`, the details of the topper in the list `l` gets stored in the variable `s`. The function `printStudentDetails()` takes a variable `s` of type `struct student` as parameter and displays all the details of student `s`. The function call `printStudentDetails(l[i])` is used to display the details of student whose details are stored in `l[i]`, the i^{th} element in the student array. We have already seen in the previous lecture, how the functions `inputStudentDetails(), printStudentDetails()` and `updateTotal()` works.

The function `getTopper()` can be declared as follows.

```
struct student getTopper(struct student [], int);
```

The function definition of the function `getTopper()` can be written as follows.

```
struct student getTopper(struct student sl[], int n)
{
   struct student s;
   int i;

   s=sl[0];
   for(i=1; i<n; i++)
   {
      if(sl[i].total > s.total)
           s=sl[i];
   }
   return s;
}
```

The above function can also be implemented without using the local structure variable `s` as given below.

```
struct student getTopper(struct student sl[], int n)
/*Get the details of the student maximum total marks
  among students in list sl of size n*/
{
   int maxpos=0, i;
   for(i=1; i<n; i++)
      if(sl[i].total > sl[maxpos].total)
          maxpos=i;
   return sl[maxpos];
}
```

As we discussed in the case of passing integer arrays as parameters, the above declaration and definition can also be written as given below.

```
struct student getTopper(struct student *, int);

struct student getTopper(struct student *sl, int n)
/*Get the details of the student maximum total marks
  among students in list sl of size n*/
{
   int maxpos=0, i;
   for(i=1; i<n; i++)
      if(sl[i].total > sl[maxpos].total)
          maxpos=i;
   return sl[maxpos];
}
```

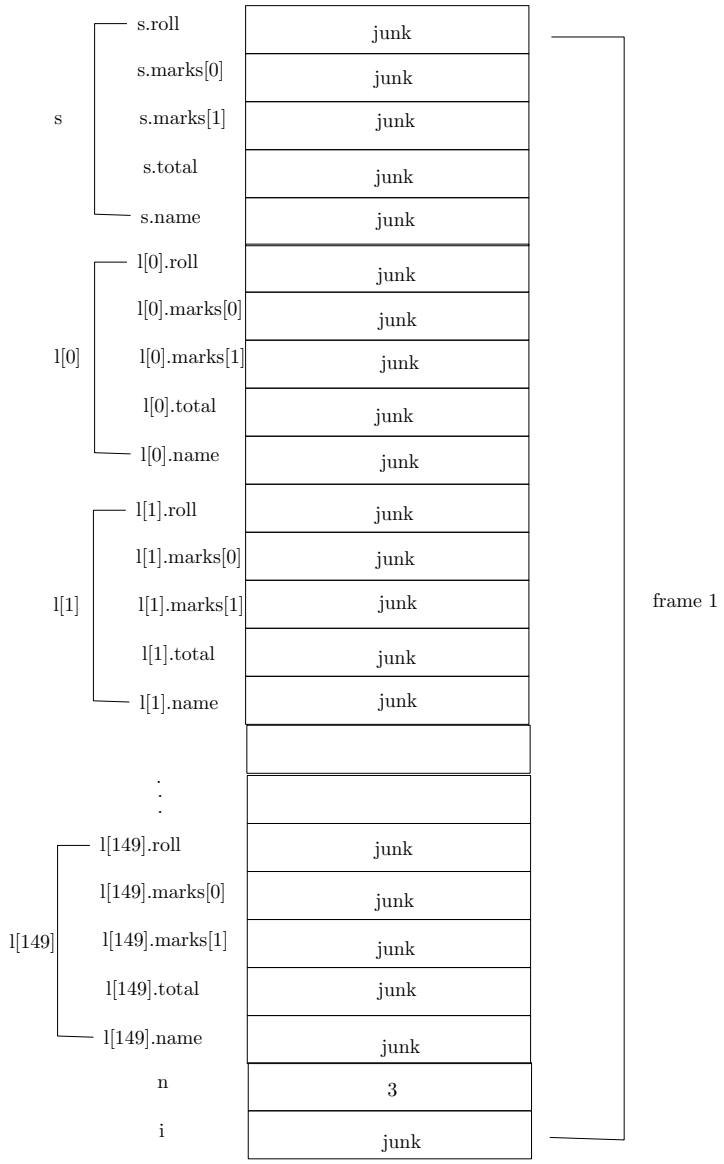
We can combine all the function definitions given above and complete the program as follows.

```
#include<stdio.h>
#define SIZE 150

struct student
{  int roll;
   unsigned int marks[2];
   unsigned int total;
   char name[15];
};

struct student inputStudentDetails();
void printStudentDetails(struct student);
void updateTotal(struct student *);
struct student getTopper(struct student [], int);

int main()
{
```

```
   struct student l[SIZE];
   struct student s;
   int i, n;
   printf("enter number of students (<%d) \n", SIZE);
   scanf("%d",&n);
   if(n > SIZE)
   {
      printf("error \n");
      return 0;
   }
   for(i=0; i<n; i++)
   {
      printf("\nGive roll number, name, mark1, mark2 of student %d\n", i+1);
      l[i]=inputStudentDetails();
      printf("\n");
      updateTotal(&l[i]);
   }
   s=getTopper(l, n);
   printf("\nDetails of the topper \n");
   printf("\n Roll No.\tName\t\tMark1\tMark2\tTotal\n");
   printStudentDetails(s);
}
struct student inputStudentDetails()
/*Input details of one student and return it */
{
      struct student s;
      scanf("%d", &s.roll);
      scanf("%14s", s.name);
      scanf("%d", &s.marks[0]);
      scanf("%d", &s.marks[1]);
      return s;
}
void printStudentDetails(struct student s)
/*Print details of student s */
{
      printf("%7d\t", s.roll);
      printf("\t%s\t", s.name);
      printf("%2d\t", s.marks[0]);
      printf("%2d\t", s.marks[1]);
      printf("%2d\n", s.total);
}
void updateTotal(struct student *ps)
/*Update the total marks of one student whose address is the parameter */
{
     (*ps).total=(*ps).marks[0]+(*ps).marks[1];
}
struct student getTopper(struct student sl[], int n)
/*Get the details of the student maximum total marks
```

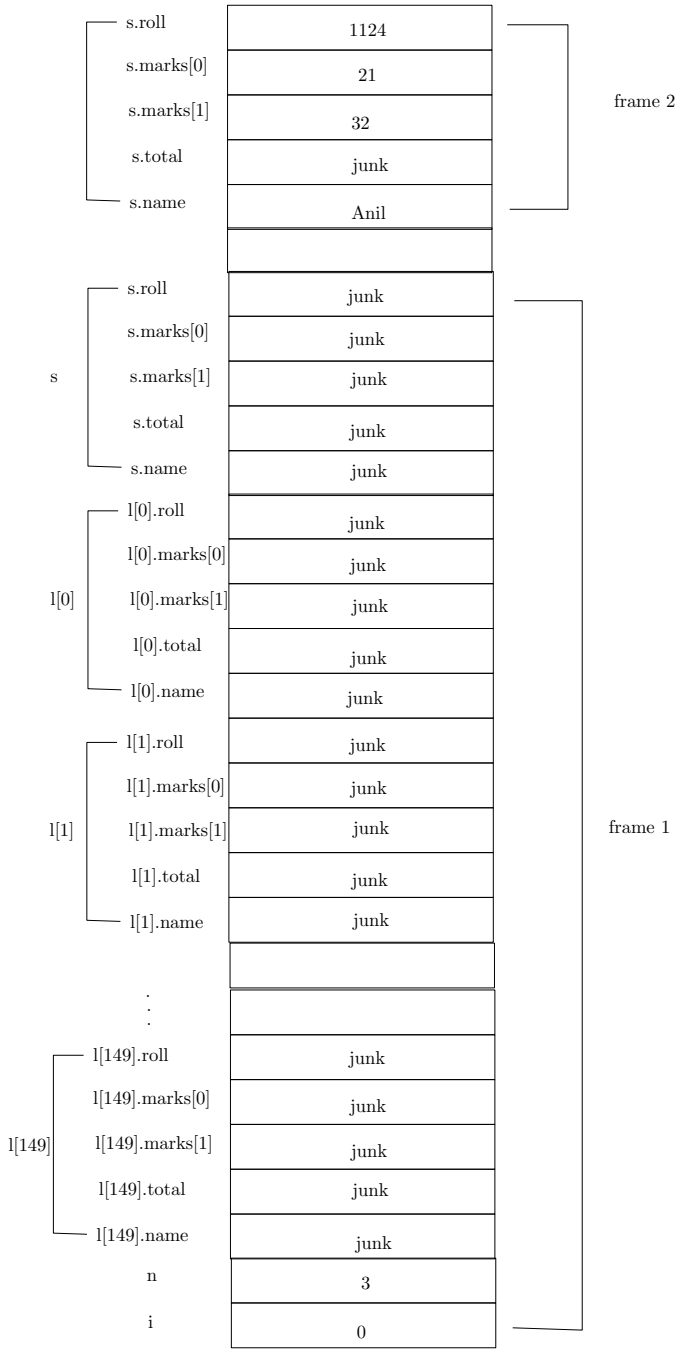
```
  among students in list sl of size n*/
{
   int maxpos=0, i;
   for(i=1; i<n; i++)
      if(sl[i].total > sl[maxpos].total)
          maxpos=i;
   return sl[maxpos];
}
```

Consider the execution of the above program. Suppose the user gives the number of students as `3`. The memory state diagram before executing the `for` loop in the `main()` function is as follows.

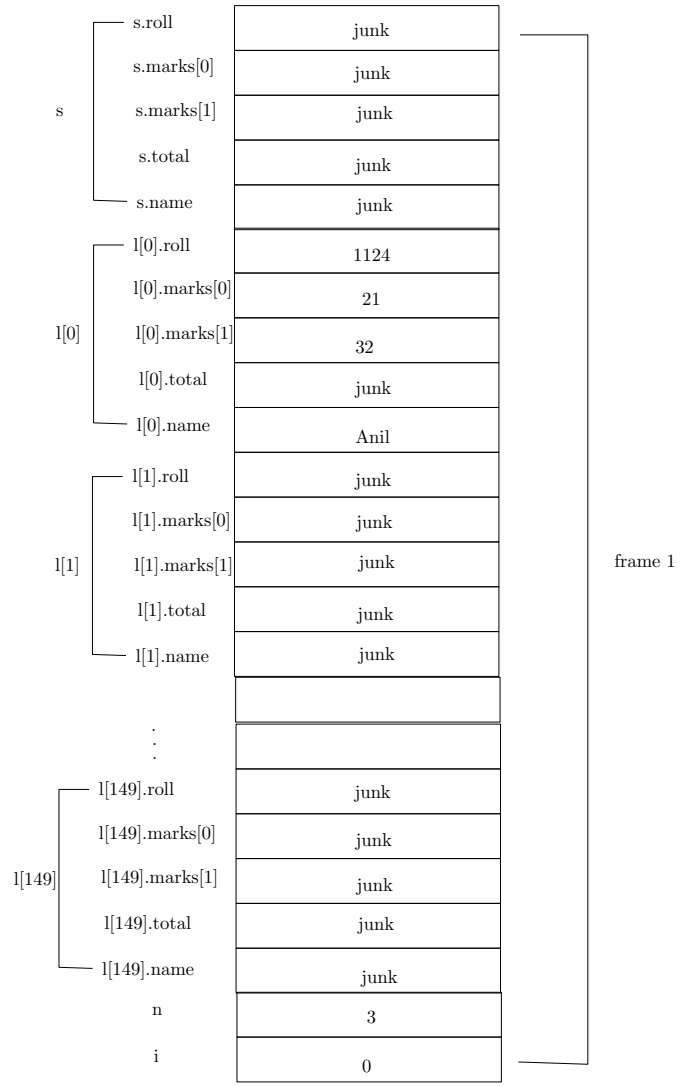


Now, the execution of the `for` loop starts. When the instruction `l[i]=inputStudentDetails();` gets executed for the first time, the function `inputStudentDetails()` gets invoked and a new frame gets created in memory.

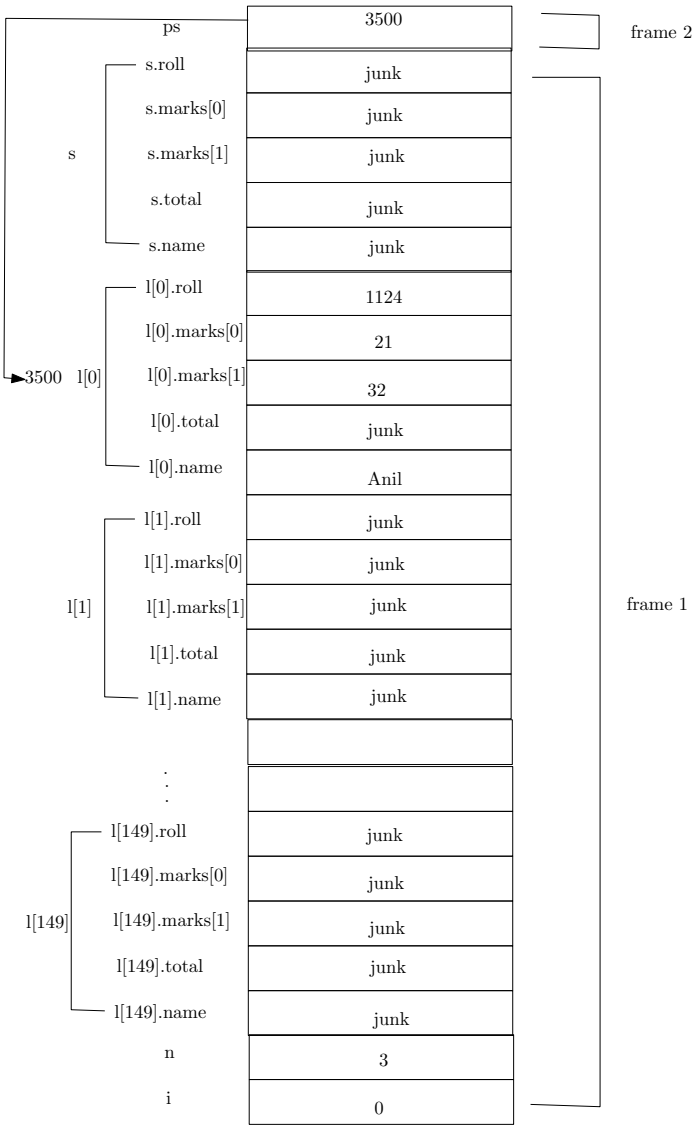
Just before finishing the execution of `inputStudentDetails()`, the memory state diagram is as follows.



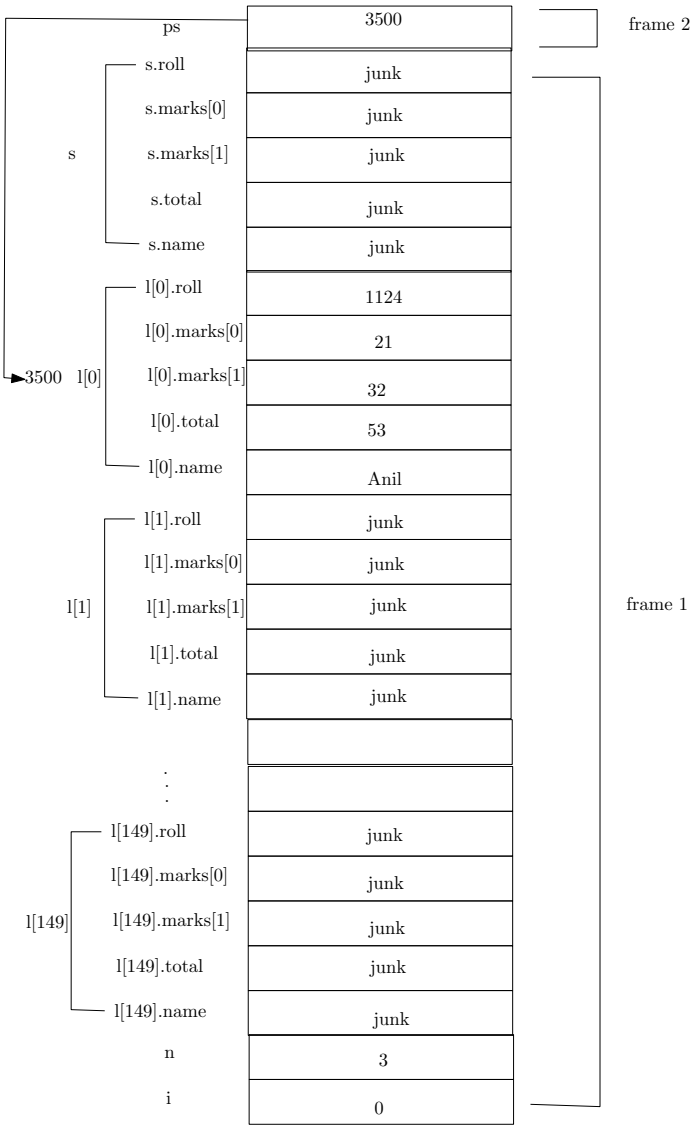
After finishing the execution of `inputStudentDetails()`, the program control returns to `main()` after copying the contents of variable `s` in frame 2 to the location of `l[0]` in frame 1.



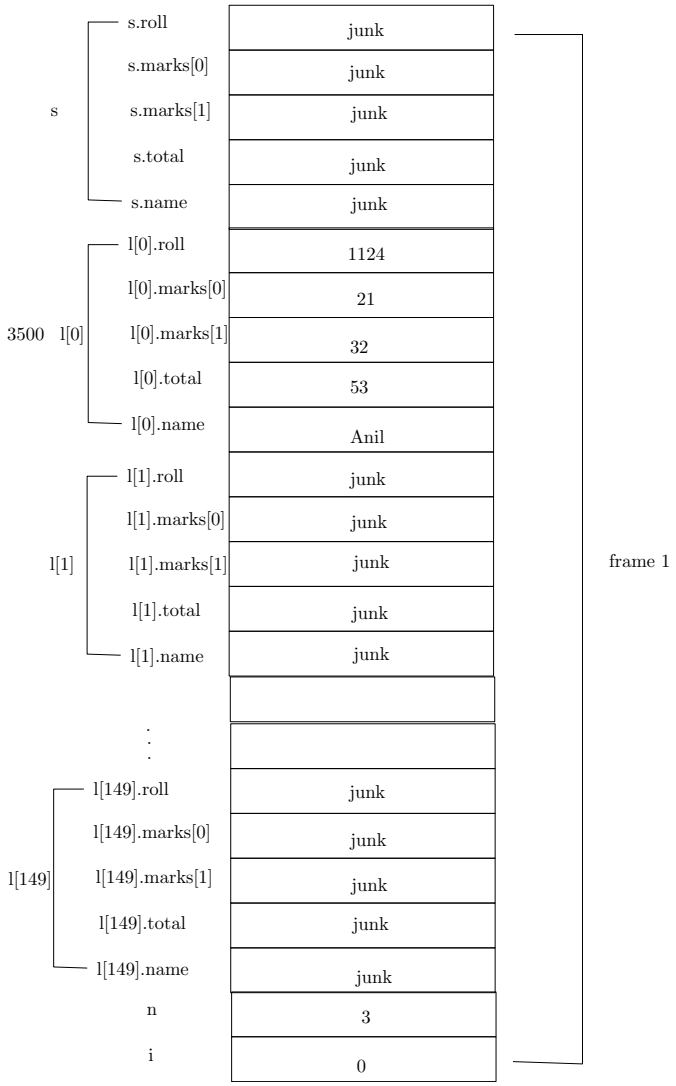
After this, the instruction `updateTotal(&l[i]);` gets executed and the function `updateTotal()` gets invoked and a new frame gets created in memory.



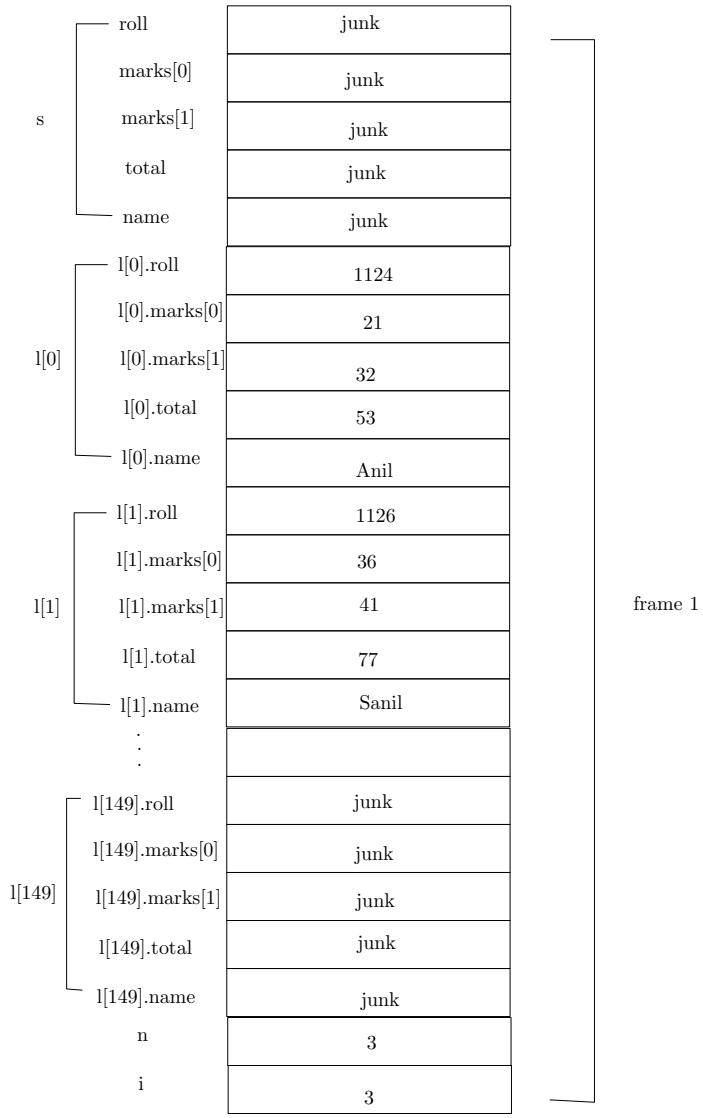
The contents of the memory just before returning from the `updateTotal()` function is as follows.



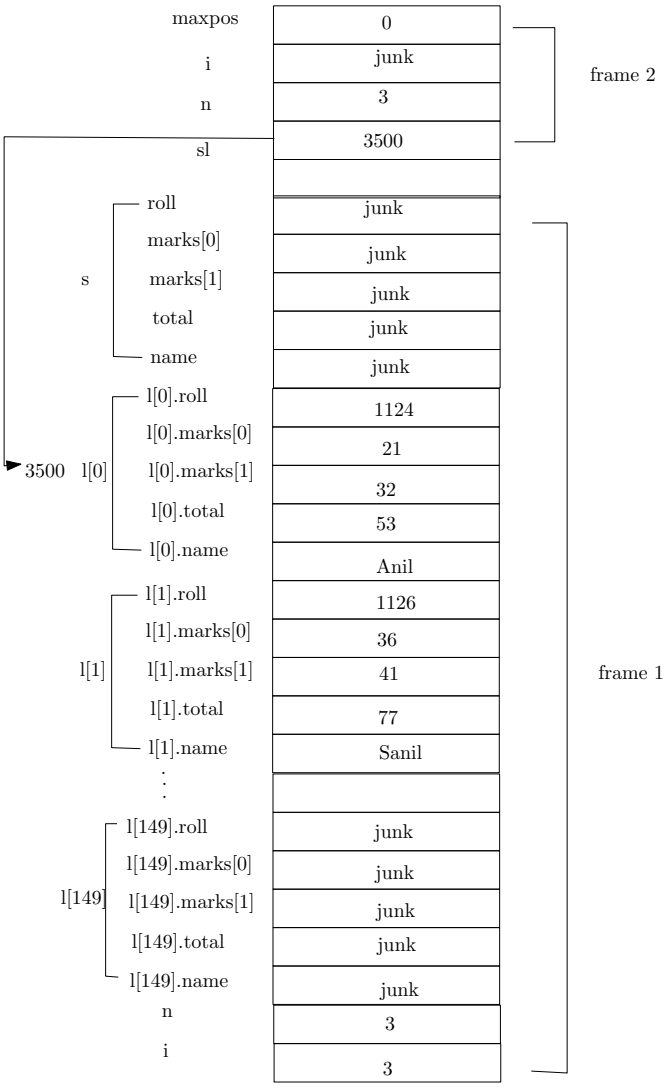
After finishing the execution of the function `updateTotal()`, frame 2 gets deleted and the control returns to `main()`. At this point, the memory state diagram is as follows.



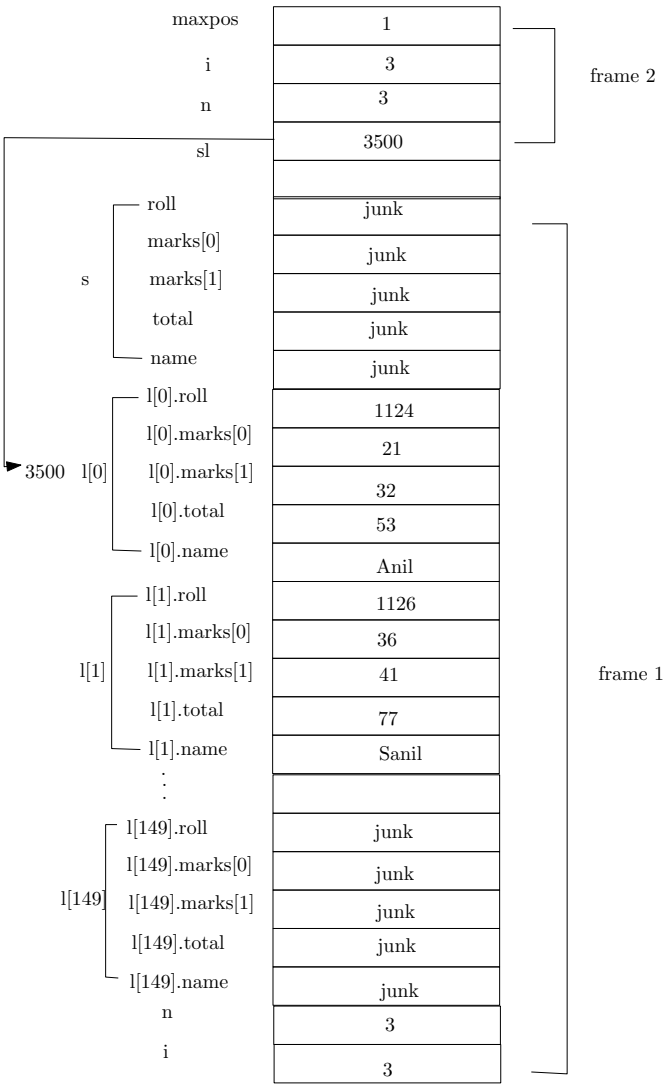
In the same way, when the `for` loop runs, the details of `3` students get stored into `l[0], l[1]` and `l[2]` and the total marks of each of these `3` students gets updated. The memory stack diagram after the execution of the above loop will be like the following.



When the function `getTopper()` is invoked, a new stack frame gets created in the memory. Just before executing the `for` loop inside the `getTopper()` function, the memory state diagram is as follows.

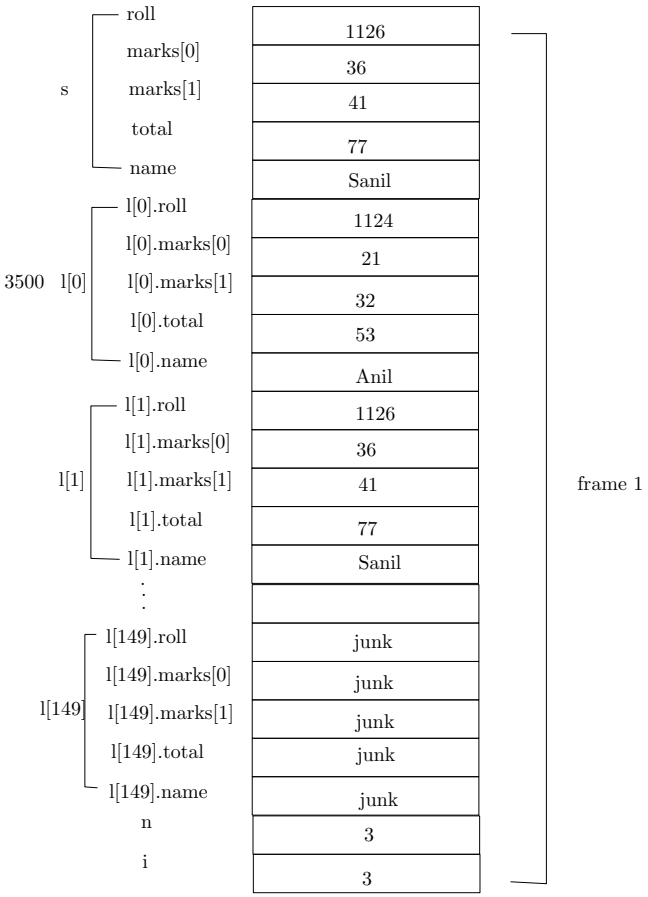


After executing the `for` loop in the `getTopper()` function, the variable `maxpos` gets the index of the student with the maximum total marks. Suppose this index is `1`. Just before returning from the function `getTopper()`, the contents of the memory is as follows.



After executing the instruction `return sl[maxpos];`, the program control returns to `main()` by copying the details of student `sl[maxpos]` (which is the same as student `sl[1]`) to the location of `s` in frame 1 and frame 2 will be destroyed.

The contents of the memory after returning from the function `getTopper()` is as follows.



After this, the function `printStudentDetails(s);` gets executed, and by doing this the details of the topper stored in variable `s` gets displayed.

Arrow operator

Suppose `ps` is a pointer to a structure variable `s` and `m` is a member field of the structure. Then, instead of writing `(*ps).m`, the short hand notation `ps->m` can be used. Using this notation, the function `updateTotal()` can be rewritten as follows.

```
void updateTotal(struct student *ps)
/*Update the total marks of one student whose address is the parameter */
{
     ps->total=ps->marks[0]+ps->marks[1];
}
```

typedef

The keyword `typedef` allows us to write short-hand representations of complex data types. Some simple examples are explained below.

If we have the definition

```
typedef int intList[15];
```

then, whenever we want to write `int x[15]`, we can write `intList x`.

If we have the definition

```
typedef int * intPtr;
```

then, whenever we want to write `int *x`, we can write `intPtr x`.

Suppose, we have a `student` structure and the instruction `struct student s[15]` declares an array of 15 elements of type `student`. We can think of giving a short-hand notation for representing an array of `15` students. A way of doing this is by using

```
typedef struct student studList[15];
```

Once we have this definition, instead of writing `struct student s[15]`, we can write `studList s[15]` . Similarly, we can define

```
typedef struct student * studPtr;
```

Once we have this, instead of writing `struct student *`, we can write `studPtr`.

The definition of a structure can be combined with a `typedef` as shown below.

```
typedef struct
{  int roll;
   unsigned int marks[2];
   unsigned int total;
   char name[15];
}student;
```

If we have this, then we can make the following definitions to give the same effect as our earlier `typedef` examples.

```
typedef student studList[SIZE];
typedef student * studPtr;
```

The following program illustrates the use of `typedef`.

```
#include<stdio.h>
#define SIZE 5

typedef struct
{  int roll;
   unsigned int marks[2];
   unsigned int total;
   char name[15];

}student;

typedef student studList[SIZE];
typedef student * studPtr;

student inputStudentDetails();
void printStudentDetails(student);
void updateTotal(studPtr);
student getTopper(studList, int);
int main()
{

   studList l;
   student s;
   int i, n;
   printf("enter number of students (<%d) \n", SIZE);
   scanf("%d",&n);
   if(n > SIZE)
   {
      printf("error \n");
      return 0;
   }
 for(i=0; i<n; i++)
   {
      printf("\nGive roll number, name, mark1, mark2 of student %d\n", i+1);
      l[i]=inputStudentDetails();
      printf("\n");
      updateTotal(&l[i]);
   }
   s=getTopper(l, n);
   printf("\nDetails of the topper \n");
   printf("\n Roll No.\tName\t\tMark1\tMark2\tTotal\n");
```

```
   printStudentDetails(s);
}
student inputStudentDetails()
{
      student s;

      scanf("%d", &s.roll);
      scanf("%14s", s.name);
      scanf("%d", &s.marks[0]);
      scanf("%d", &s.marks[1]);
      return s;
}
void printStudentDetails(student s)
{

      printf("%7d\t", s.roll);
      printf("\t%s\t", s.name);
      printf("%2d\t", s.marks[0]);
      printf("%2d\t", s.marks[1]);
      printf("%2d\n", s.total);

}
void updateTotal(studPtr ps)
{
     ps->total=ps->marks[0]+ps->marks[1];
}
student getTopper(studList sl, int n)
{
   int maxpos=0, i;
   for(i=1; i<n; i++)
      if(sl[i].total > sl[maxpos].total)
          maxpos=i;
   return sl[maxpos];
}
```