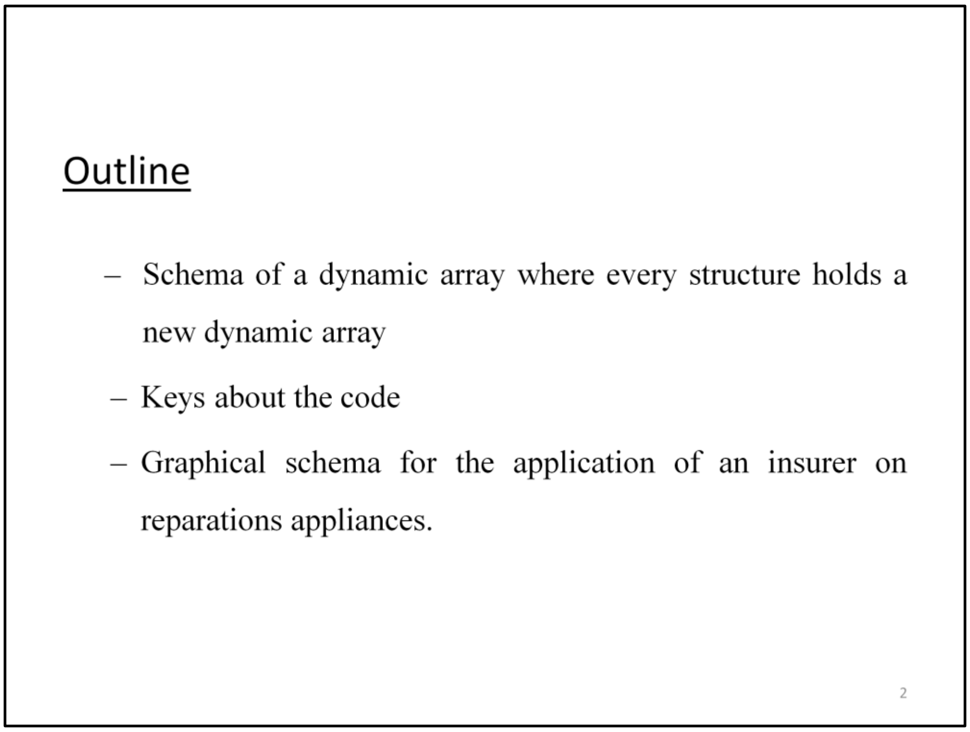
MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Pointers in C Language

Concha Batanero Ochaíta *(concha.batanero@uah.es)*

Helsinki Metropolia University of Applied Sciences

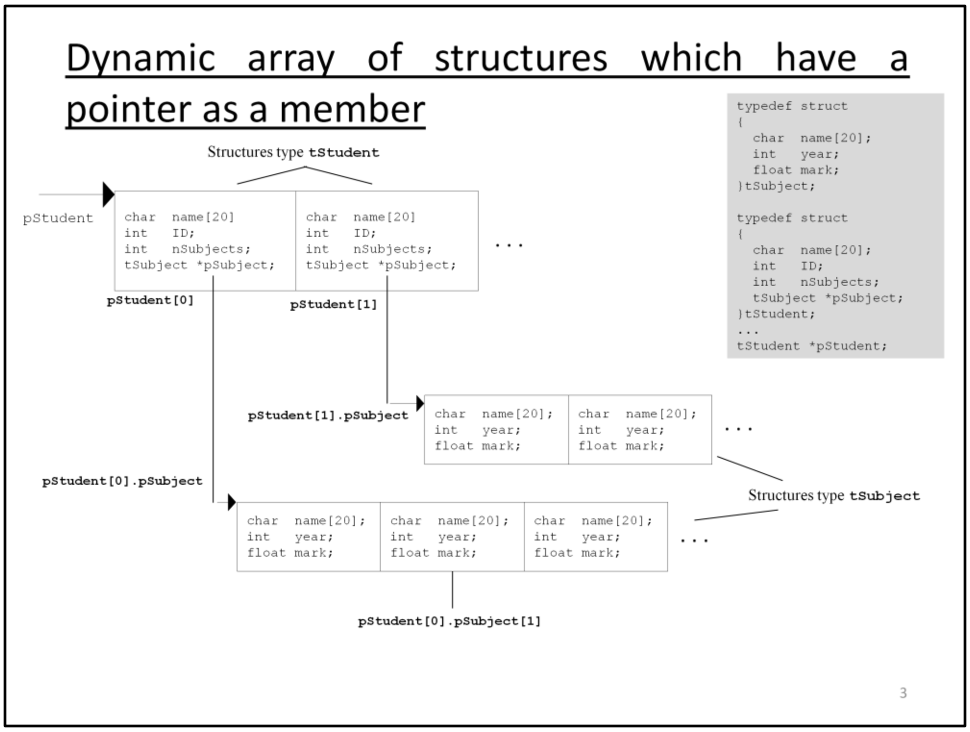


Outline

– Schema of a dynamic array where every structure holds a new dynamic array

– Keys about the code

– Graphical schema for the application of an insurer on reparations appliances.



Dynamic array of structures which have a pointer as a member

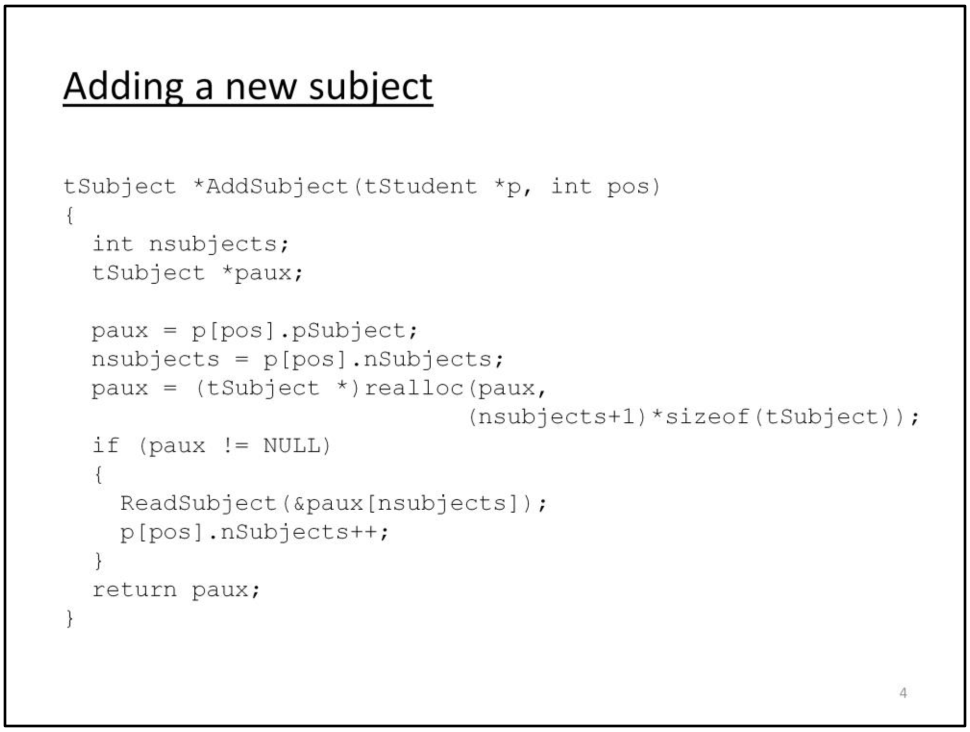
```
typedef struct
{
  char  name[20];
  int   year;
  float mark;
}tSubject;

typedef struct
{
  char  name[20];
  int   ID;
  int   nSubjects;
  tSubject *pSubject;
}tStudent;
...
tStudent *pStudent;
```

This slide represents the case that one member of the structure of a dynamic array is a pointer. We have a structure that contains the personal data of a student: name, ID and number of subjects that he or she is enrolled. Moreover, the structure contains a pointer to the structure *tSubject* which stores the next information of a subject: name, year of enrollment and the obtained mark. It means that every student structure has a pointer pointing to a new dynamic array with the data of their subjects. So, we have one dynamic array pointed by *pStudent* which stores the information of the students and one dynamic array for every student which stores the data of the subjects of the student. We must create all of them. Of course we must start for the students array. The size of the dynamic arrays of subjects is determined by the member *nSubjects* located in the structure *tStudent*.

As you can see, the image clarifies the explanation. It is part very important to successful finish the exercise. Otherwise we can finish lost between the different pointers and structures. Therefore, from now we are going to draw a schema of the organization of the dynamic arrays to a better following of the data. It is necessary since the complexity is increasing.

We are going to program an application to manage these dynamic arrays. The program offers a menu where the users are able to add a student and a subject every time that they press the options *1* or *2*.

The *realloc* function will be used in this occasion to create the dynamic arrays. This function creates the first element of the array if a pointer pointing to *NULL* is passed as one of its parameters. Otherwise, if the pointer received by *realloc* is pointing to a dynamic array, already created, a new element will be added at the end of the array.



Adding a new subject

```
tSubject *AddSubject(tStudent *p, int pos)
{
  int nsubjects;
  tSubject *paux;

  paux = p[pos].pSubject;
  nsubjects = p[pos].nSubjects;
  paux = (tSubject *)realloc(paux,
                            (nsubjects+1)*sizeof(tSubject));
  if (paux != NULL)
  {
    ReadSubject(&paux[nsubjects]);
    p[pos].nSubjects++;
  }
  return paux;
}
```

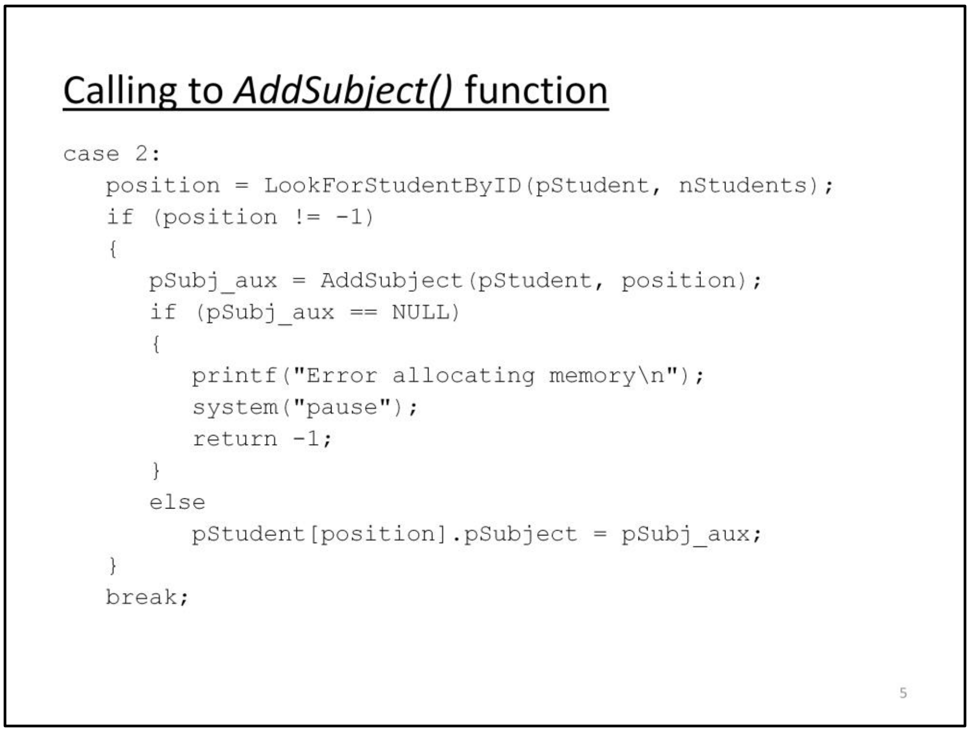
4

Let see the function *AddSubject* that adds a subject to a specific student.

Every time the user selects the option *2* we should add a new subject at the end of the subjects array of one specific student. The function receives the pointer to the beginning of the students array and the position in the students array of the student whose subject is going to be added. The pointer pointing to the subjects array of one student, which position is *pos*, is the following:

pStudent[pos].pSubject

The steps to allocate dynamic memory for a new subject are the same we showed before: reading the data, increasing of the numbers of subjects, etc.



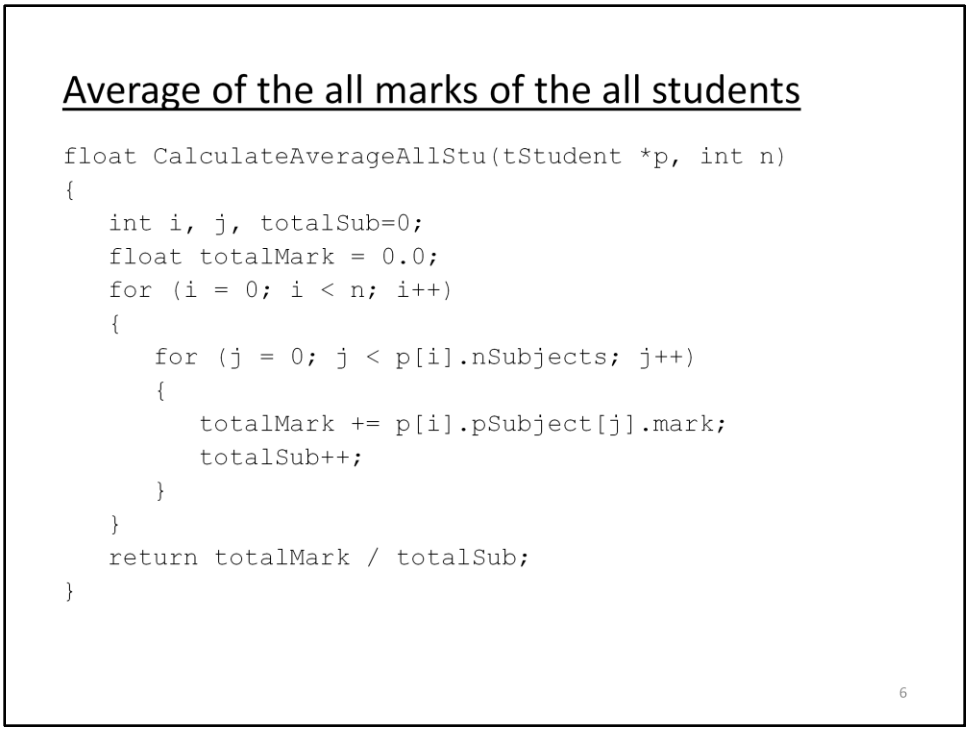
Calling to *AddSubject()* function

```
case 2:
   position = LookForStudentByID(pStudent, nStudents);
   if (position != -1)
   {
      pSubj_aux = AddSubject(pStudent, position);
      if (pSubj_aux == NULL)
      {
         printf("Error allocating memory\n");
         system("pause");
         return -1;
      }
      else
         pStudent[position].pSubject = pSubj_aux;
   }
   break;
```

5

Here is the code fragment of the *main()* function that shows the case *2* responsible for the adding of a new subject to a student. Before adding the subject, we should ask the ID of the student whose subject must be added and look for the position of this student in the array of students. The subject will not be added if the student doesn’t exist.

Notice that the function *LookForStudentByID()* returns either the position of the student in the students array or the value *-1* if the ID that has been read from keyboard doesn’t exist in the students arrays.

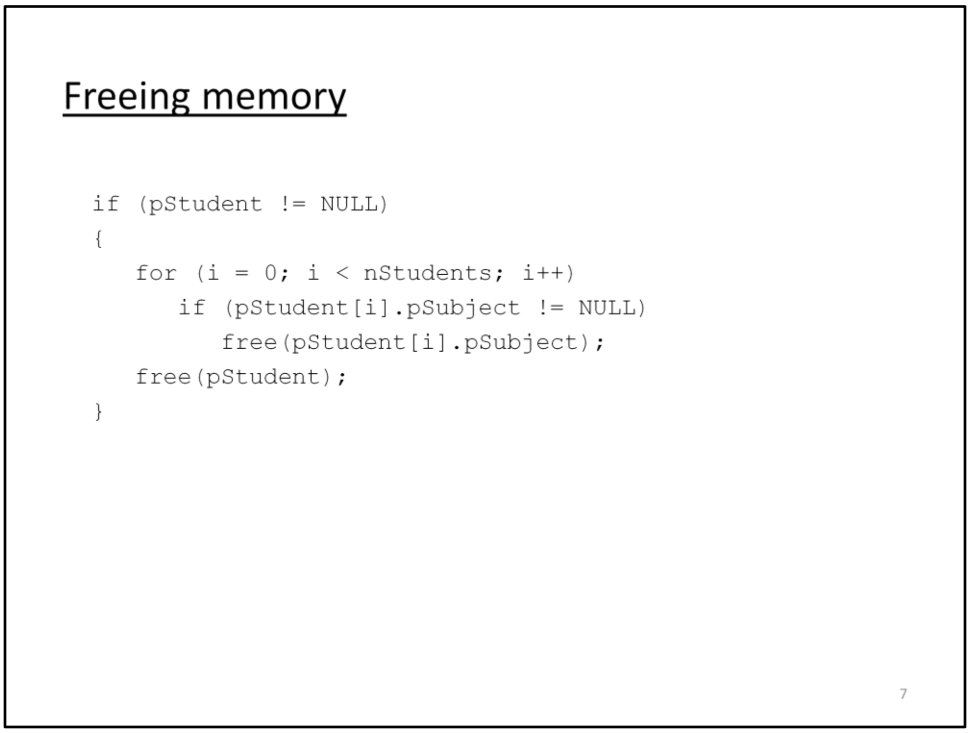


Average of the all marks of the all students

```
float CalculateAverageAllStu(tStudent *p, int n)
{
   int i, j, totalSub=0;
   float totalMark = 0.0;
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < p[i].nSubjects; j++)
      {
         totalMark += p[i].pSubject[j].mark;
         totalSub++;
      }
   }
   return totalMark / totalSub;
}
```

6

Once we have built the dynamic arrays we can operate with the data and select those we need. For example, the all marks of the all subjects of every student should be added, as we can see on the slide, to calculate the total average of the marks.



Freeing memory

```
if (pStudent != NULL)
{
   for (i = 0; i < nStudents; i++)
      if (pStudent[i].pSubject != NULL)
         free(pStudent[i].pSubject);
   free(pStudent);
}
```

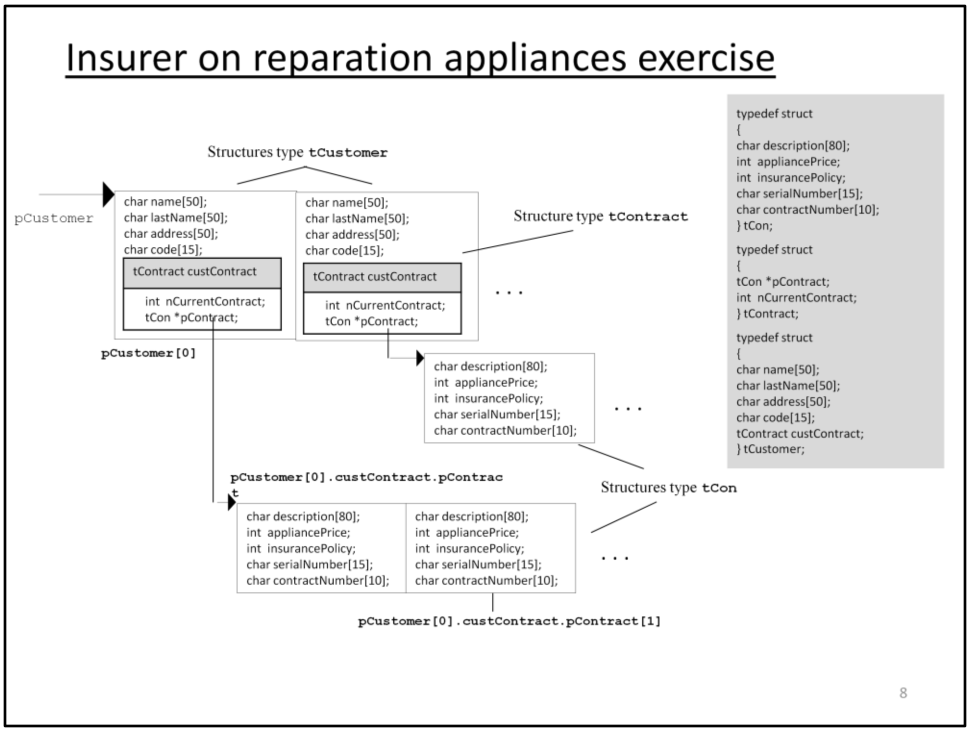
7

Other aspect which deserves attention is the way to free the memory. In this case there are several pointers to be freed, not just one. We should free all of them. First we go along the students array and for each student we liberate his subjects array through this statement:

free(pStudent[i].pSubject);

After that, we are allowed to free the pointer of the students array.
We should do in this order and no in the opposite order, because we need the pointer *pStudent* to access the subjects. If we free this pointer first, then we will not be able to access the subject pointers to free them and these memories areas will be lost.

Notice that we check if the pointer is different of *NULL* before free it. As we said it is not allowed to free pointers that point to *NULL*. This statement will avoid some run-time errors.



Insurer on reparation appliances exercise

```
typedef struct
{
char description[80];
int appliancePrice;
int insurancePolicy;
char serialNumber[15];
char contractNumber[10];
} tCon;

typedef struct
{
tCon *pContract;
int nCurrentContract;
} tContract;

typedef struct
{
char name[50];
char lastName[50];
char address[50];
char code[15];
tContract custContract;
} tCustomer;
```

8

This slide offers help to the second exercise.

The only difference of this schema with the schema of the previous application is that inside of every structure of the customers array, there is other structure which holds the number of contracts and the pointer to the contracts.