Self referential data structures

In this document I will speak about one of the data structures that are said to be *self referential*.

Linked lists and *stacks* belong to the mentioned kind of data structures. This kind of data structures can be created in a way that in a structure we declare a pointer that points to another structure whose type is the same as the type of the one that involves the pointer itself.

After declaring some structures (more precisely: objects) from this kind, each of them can be concatenated into a link by using the pointers that were mentioned above.

It is important for the linked structures / objects to belong to the same type (to have the same inner structure). In this way we get a singly linked list. Let us see how a stack would look like in case we stored data of the racers of *Hogwarts School of Witchcraft and Wizardry* by using a stack.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>


typedef struct cup
{
 char name[32];
 int class;
 int score;
 struct cup *next;
}cup;


cup *first = NULL;


int push(char *n, int c, int s);
int pop();


int main(){
 char name[32], c; int class, score;

 printf("\nFilling the stack:\n");
```

```
 do
 {
  printf("\nName: "); scanf("%s", name);
  printf("Class: "); scanf("%d", &class);
  printf("Score: "); scanf("%d", &score);

  if(push(name, class, score))
  {
   printf("Unsuccessful allocating."); return 1;
  }

  printf("\nIs there any other student? (y / n) ");
  scanf(" %c", &c);
  system("clear"); /*clearing the console, (if you use
  Windows: system("cls")   )*/
 }while(c == 'y');

 printf("\n\nPrinting and clearing the stack:\n\n");

 while(!pop());

 return 0;
}


int push(char *n, int c, int s)
{
 cup *newElement; int i = 0;
 if(!(newElement = (cup *)malloc(sizeof(cup)))){return 1;}
 strcpy(newElement->name, n);
 newElement->class = c;
 newElement->score = s;
 newElement->next = first;
 first = newElement;
 return 0;
}


int pop()
{
 cup *oldElement;
 if(first == NULL){return 1;}
 oldElement = first;
 printf("\nname: %s\t", first->name);
 printf("class: %d\t", first->class);
 printf("score: %d\n", first->score);
 first=first->next;
 free(oldElement);
 return 0;
}
```

To understand how this program works, we have to remember what we learned of the *stack*. It is essential to know that constructing the pointer that points to the other structure / object, in the following code

```
typedef struct cup
{
 char name[32];
 int class;
 int score;
 struct cup *next;
}cup;
```

the name of `cup` cannot be used to give the data type that will be pointed to by the pointer. At this time only `struct cup` is allowed to be used.

Running the program we can see that the elements that we have written are printed in a reverse sequence.

The last one is printed first and the first one is the last that is printed. The reason why we can see it is that writing and reading are done at the same end of the link. (Naturally, we can consider the link as a vertical one. In this case the end of the link becomes the top of the stack.) So, the data that was last pushed in is pop out first.