Implementing Array as an Abstract Data Type in C Language

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-8

In the last tutorial, we discussed the blueprint of our customized abstract data type, myArray. In case you missed it, make sure you check it out. Today, we will learn to write the code to implement that array with all the previously defined sets of values and operations.

Editor settings:

I will recommend you to use MinGW w64-bit compiler to compile your C programs and VS Code as your code editors. VS Code is highly recommended for its versatility with all the programming languages in the market. You can even check out my Youtube video covering all of this. Let's, for now, assume that you all have your setup ready. I have attached the code snippet for creating the above ADT array below. Let's check it out.

Understanding the snippet below:

1. First, we will define a structure. You can use a class and its methods in C++, but in C, a structure is used to define customized data types.
2. Keep the blueprint we made in the last tutorial by your side. Define the structure elements, integer variables total_size and used_size, and an integer pointer to point at the address of the first element.
3. We are now ready with our customized data type. Let's define some functions, which will feature

- Creating an array of this data type,
- Printing the contents of this array,
- Setting values in this array.

Create a void function *createArray* by passing the address of a struct data type *a*, and integers tSize and uSize. We can very easily assign this tSize and uSize given from the main, to the total_size and used_size of the struct myArray *a* by either of the methods defined below.

```
(*a).total_size = tSize;
            or
 a->total_size = tSize;
```

Code Snippet 1: Syntax for assigning structure elements to structure pointers.

Similarly, assign the integer pointer ptr, the address of the reserved memory location using malloc. Do use the header file <stdlib.h> for using malloc.

```
a->ptr = (int *)malloc(tSize * sizeof(int));
```

Code Snippet 2: Using malloc

4. We will now create a *show* function to display all the elements of the struct myArray. We will simply pass the address of the struct myArray *a*. To print all the elements, we will traverse through the whole struct and print each struct element till the iterator reaches the last element. We will use *a→used_size* to define the loop size. Use (*a→ptr)[i]* to access each element.

5. We will now create a setVal function to set values to this struct myArray a and pass the address of the same. Use scanf to assign values to each element via (a→ptr)[i] .

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

struct myArray
{
    int total_size;
    int used_size;
    int *ptr;
};

void createArray(struct myArray * a, int tSize, int uSize){
    // (*a).total_size = tSize;
    // (*a).used_size = uSize;
    // (*a).ptr = (int *)malloc(tSize * sizeof(int));

    a->total_size = tSize;
    a->used_size = uSize;
    a->ptr = (int *)malloc(tSize * sizeof(int));
}

void show(struct myArray *a){
    for (int i = 0; i < a->used_size; i++)
    {
        printf("%d\n", (a->ptr)[i]);
    }
}

void setVal(struct myArray *a){
    int n;
    for (int i = 0; i < a->used_size; i++)
    {
        printf("Enter element %d", i);
        scanf("%d", &n);
        (a->ptr)[i] = n;
    }

}

int main(){
    struct myArray marks;
    createArray(&marks, 10, 2);
    printf("We are running setVal now\n");
    setVal(&marks);

    printf("We are running show now\n");
    show(&marks);

    return 0;
}
```

Code Snippet 3: A program to implement the ADT array

So, these were the basic methods we could define for this struct. We'll check if these work by running it. We'll call the *createArray*, and *setVal* functions first to create an array of size 2, and assign some values to it. And then call the show function to see if it works.

Output of the above program:

```
Enter element 0 : 12
Enter element 1 : 13
We are running show now
12
13
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

And this was implementing the myArray ADT. I hope you all could follow it. Possibly there were some syntaxes you were not familiar with, but don't worry, take your time. Watch the other courses regarding them on my Youtube channel.

Thank you for being with me throughout. I hope you enjoyed the tutorial. If you appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll learn to operate on this array using several operators. Till then keep learning.

Here is the source code we wrote in the video!

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

struct myArray
{
    int total_size;
    int used_size;
    int *ptr;
};

void createArray(struct myArray * a, int tSize, int uSize){
    // (*a).total_size = tSize;
    // (*a).used_size = uSize;
    // (*a).ptr = (int *)malloc(tSize * sizeof(int));

    a->total_size = tSize;
    a->used_size = uSize;
    a->ptr = (int *)malloc(tSize * sizeof(int));
}

void show(struct myArray *a){
    for (int i = 0; i < a->used_size; i++)
    {
        printf("%d\n", (a->ptr)[i]);
    }
}

void setVal(struct myArray *a){
    int n;
    for (int i = 0; i < a->used_size; i++)
    {
        printf("Enter element %d", i);
        scanf("%d", &n);
        (a->ptr)[i] = n;
    }

}

int main(){
    struct myArray marks;
    createArray(&marks, 10, 2);
    printf("We are running setVal now\n");
    setVal(&marks);

    printf("We are running show now\n");
    show(&marks);

    return 0;
}
```