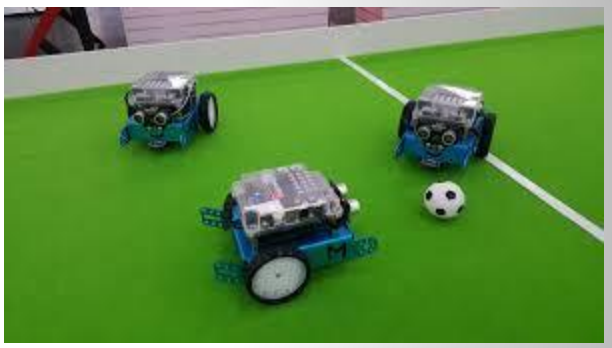
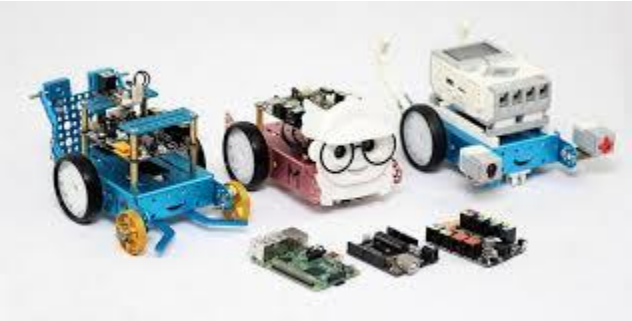
Arrays

Problema

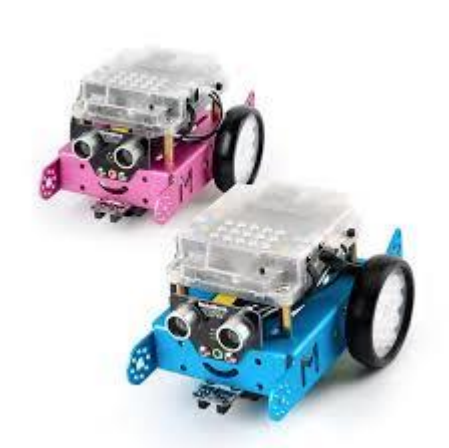


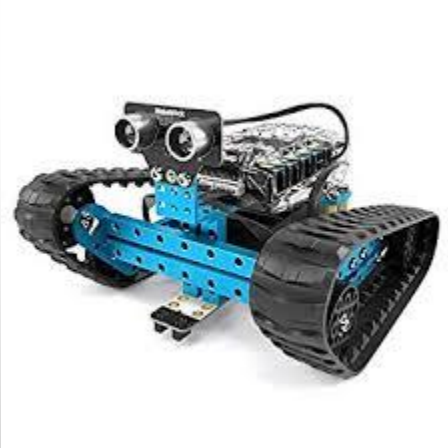


Cómo catalogar mis mBot?

Posible Solución

```
string blue_mbot_1;
string blue_mbot_2;
string pink_mbot_1;
…
string blue_mbot_ranger_1;
```





Solución mejor: Arrays

```
const short MBOTS_SIZE = 18;
string my_mbots[MBOTS_SIZE];
```

Declaración de arrays

- Sintaxis:

```
base_type array_name[number_of_cells];

// examples
float student_grades[50];
string student_names[50];
int student_ids[50];

const short NUM_FRIENDS = 27;
string my_friends_names[NUM_FRIENDS];
```

Ejemplo

```
const int SAMPLE_SIZE = 340;
float ultrasound_readings[SAMPLE_SIZE];
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
3.56	3.44	4.03	3.96	3.77	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
ultrasound_readings[0] = 45.1;
ultrasound_readings[4] = 12.1;
cout << ultrasound_readings[6] << endl;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
3.56	3.44	4.03	3.96	3.77	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
ultrasound_readings[0] = 45.1;
ultrasound_readings[4] = 12.1;
cout << ultrasound_readings[6] << endl;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
3.56	3.44	4.03	3.96	3.77	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
ultrasound_readings[0] = 45.1;
ultrasound_readings[4] = 12.1;
cout << ultrasound_readings[6] << endl;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
45.1	3.44	4.03	3.96	3.77	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
ultrasound_readings[0] = 45.1;
ultrasound_readings[4] = 12.1;
cout << ultrasound_readings[6] << endl;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
45.1	3.44	4.03	3.96	3.77	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
ultrasound_readings[0] = 45.1;
ultrasound_readings[4] = 12.1;
cout << ultrasound_readings[6] << endl;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
45.1	3.44	4.03	3.96	12.1	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
ultrasound_readings[0] = 45.1;
ultrasound_readings[4] = 12.1;
cout << ultrasound_readings[6] << endl;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
45.1	3.44	4.03	3.96	12.1	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
ultrasound_readings[0] = 45.1;
ultrasound_readings[4] = 12.1;
cout << ultrasound_readings[6] << endl;
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
45.1	3.44	4.03	3.96	12.1	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
const int SAMPLE_SIZE = 340;
float ultrasound_readings[SAMPLE_SIZE];
...
cout << ultrasound_readings[SAMPLE_SIZE];
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]
45.1	3.44	4.03	3.96	12.1	3.49	3.92	…	4.01	3.83	3.21

Acceso a elementos

```
const int SAMPLE_SIZE = 340;
float ultrasound_readings[SAMPLE_SIZE];
...
cout << ultrasound_readings[SAMPLE_SIZE];
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	…	[337]	[338]	[339]	[340]
45.1	3.44	4.03	3.96	12.1	3.49	3.92	…	4.01	3.83	3.21	?

Nos salimos del array !!! - Acceso a una zona de memoria “peligrosa”

Inicialización de arrays

```
const int SIZE = 5;
int array1[SIZE] = {7,2,4,5,1};
```

[0]	[1]	[2]	[3]	[4]
7	2	4	5	1

Inicialización de arrays

```
const int SIZE = 5;
int array2[SIZE] = {0};
```

[0]	[1]	[2]	[3]	[4]
0	0	0	0	0

Inicialización de arrays

```
const int SIZE = 5;
int array3[SIZE] = {9,1};
```

[0]	[1]	[2]	[3]	[4]
9	1	0	0	0

Inicialización de arrays

```
int array4[] = {5,2,4};
```

[0]	[1]	[2]
5	2	4

Lo que NO debemos hacer

- Hacer “return” de un array en una función

```
int[] function(void)
{
    int arrayf[] = {1,2,3};
    return arrayf;
}
```

Lo que NO debemos hacer

- Hacer “return” de un array en una función
- Imprimir un array como si fuera un tipo básico

```
int array1[10];
cout << array1 << endl;
```

Lo que NO debemos hacer

- Hacer “return” de un array en una función
- Imprimir un array como si fuera un tipo básico

```
int array1[10];
cout << array1 << endl;
```

- Leer un array completo de la entrada

```
int array2[10];
cin >> array2;
```