FUNCTIONS IN C

Objectives

To learn the relationship of pointers and arrays.

To access arrays through pointers.

What are STRUCTURED DATA TYPES*?*

The collection of simple data type values…

…arranged in some manner to facilitate easier access.

e.g. arrays, strings,
and structures

ARRAYS

One-dimensional arrays

A collection of data of the same type.

Referenced by a common name or identifier.

```
<data_type> <var_name>[size];
```

```
<data_type> <var_name>[size];
```

ANY VALID TYPE

```
<data_type> <var_name>[size];
```

ANY VALID IDENTIFIER

```
<data_type> <var_name>[size];
```

THE MAXIMUM NUMBER OF ELEMENTS

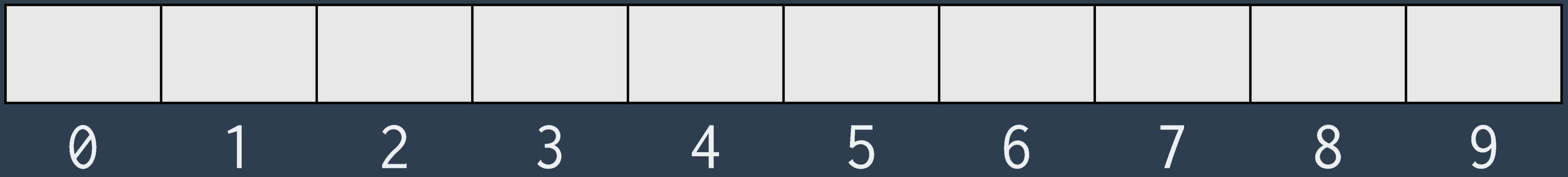
```
int numbers[10];
float grade[50];
```

Arrays in the memory

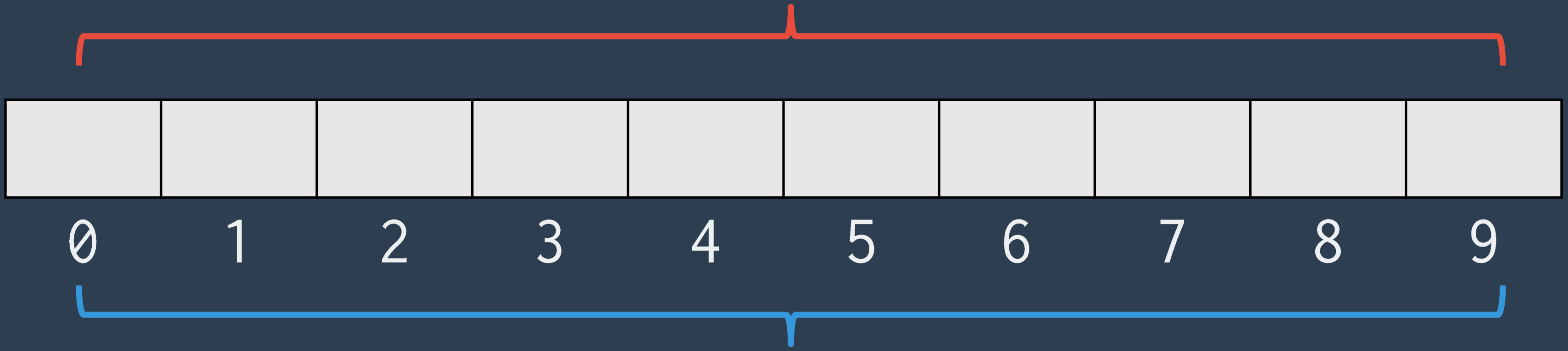
When an array is declared,…

…consecutive memory locations are reserved.

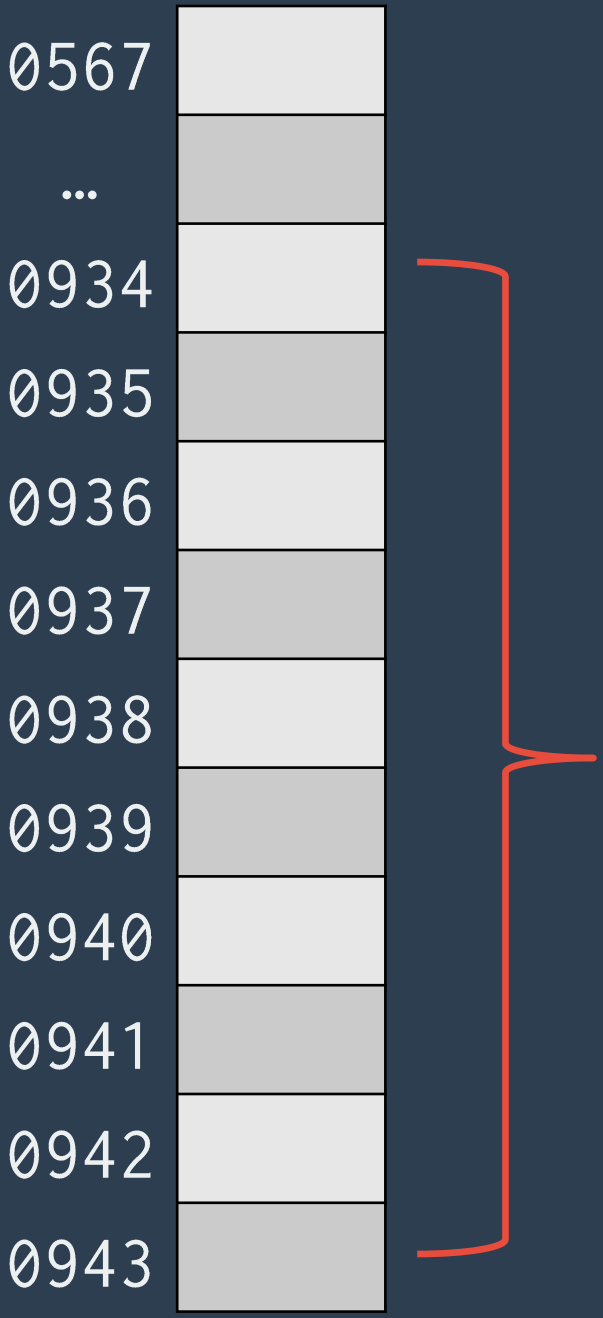
```
int num[10];
```


0
1
2
3
4
5
6
7
8
9

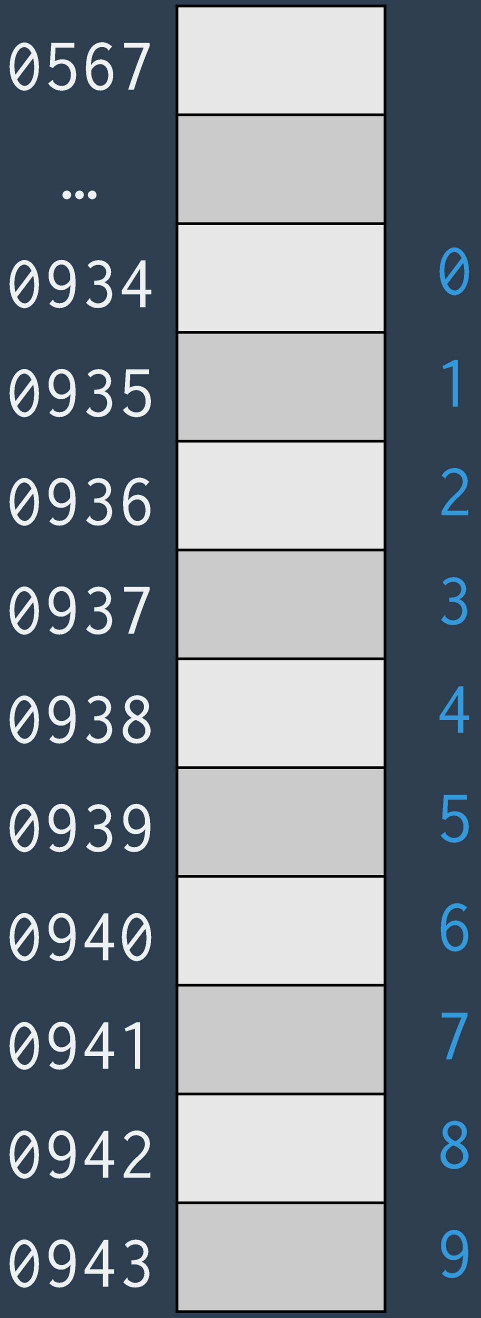
ARRAY ELEMENTS



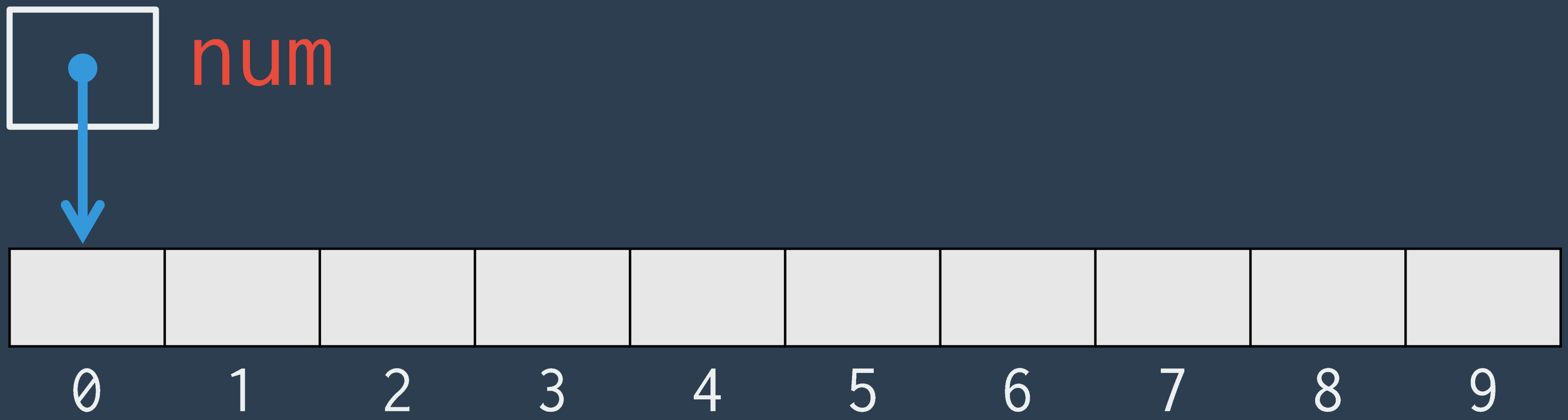
ARRAY INDEX

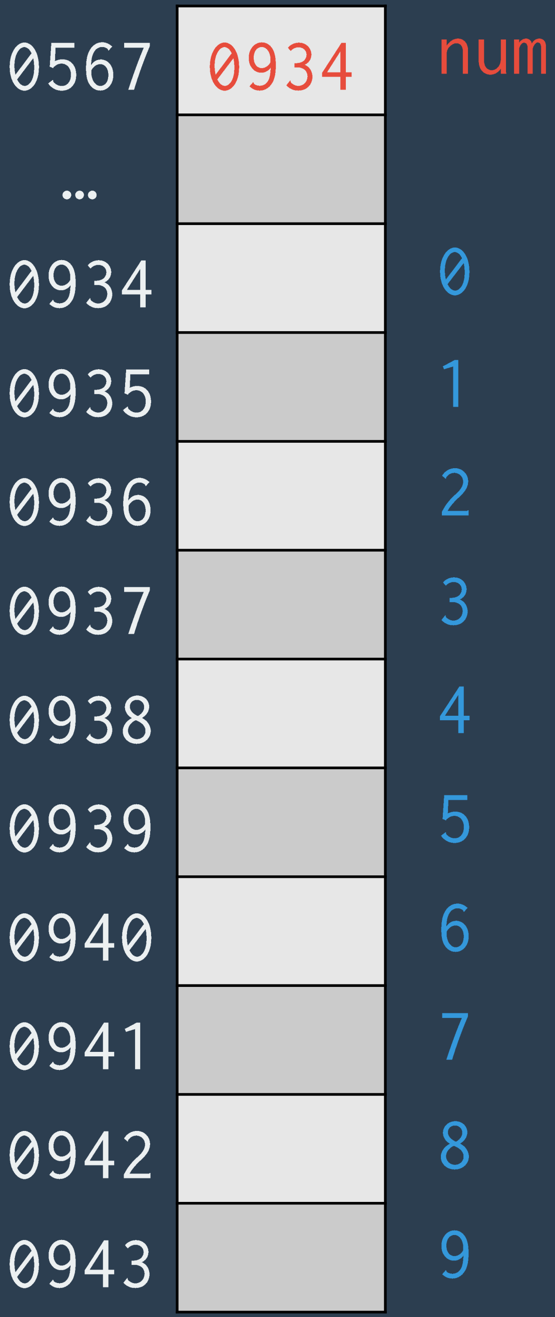


ARRAY ELEMENTS


0567
...
0934
0935
0936
0937
0938
0939
0940
0941
0942
0943
0
1
2
3
4
5
6
7
8
9

The variable name is a (constant) pointer to the first element of the array.


num
0
1
2
3
4
5
6
7
8
9



0567	0934	num
…		
0934		0
0935		1
0936		2
0937		3
0938		4
0939		5
0940		6
0941		7
0942		8
0943		9

Total space allocated for an array:

Consecutive memory locations equivalent to size

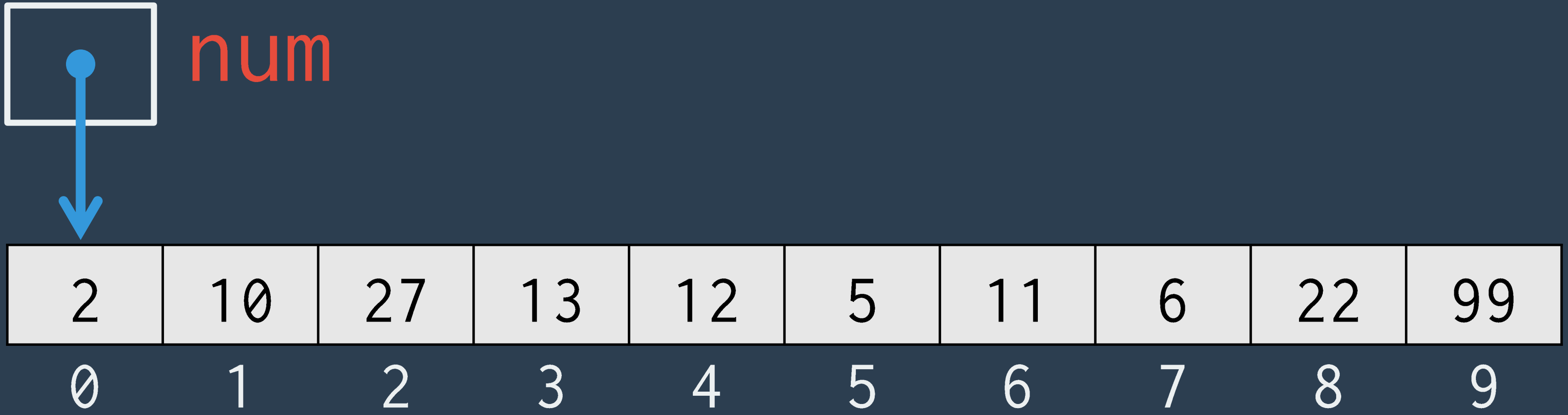
+

A space for the pointer to the first element

Initializing arrays

An array may be **initialized** during declaration.

```
int num[10] = {2, 10, 27, 13,
        12, 5, 11, 6, 22, 99};
```



2	10	27	13	12	5	11	6	22	99
0	1	2	3	4	5	6	7	8	9

Accessing arrays

Two ways of accessing array elements:

Indexing

+

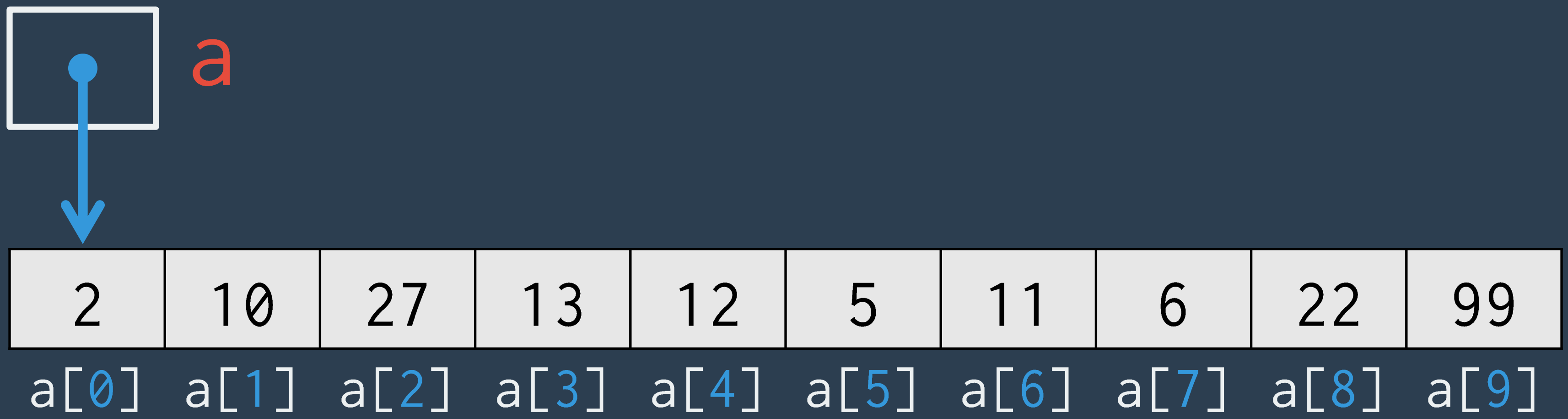
Pointer arithmetic

INDEXING

Arrays are numbered from 0 to size-1.

The first element is index 0;

the last is index size-1.



2	10	27	13	12	5	11	6	22	99
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

```
<var_name>[index]
```

```
num[8] = 10;
```

When accessing array components, make sure that the index is within the bounds of the array.

```
int a[10];

a[-1] = 0; //invalid?
a[10] = 9; //invalid?
a[4] = 3;  //valid
```

Only integers are allowed as index.

```
int a[10];
int i=2, j=1;

a[7.8] = 0;
a[j/i] = 23;
a[(i*j)%3] = 34;
```

```
int a[10];
int i=2, j=1;

a[7.8] = 0; //invalid
a[j/i] = 23; //stores in a[0]
a[(i*j)%3] = 34; //stores in a[2]
```

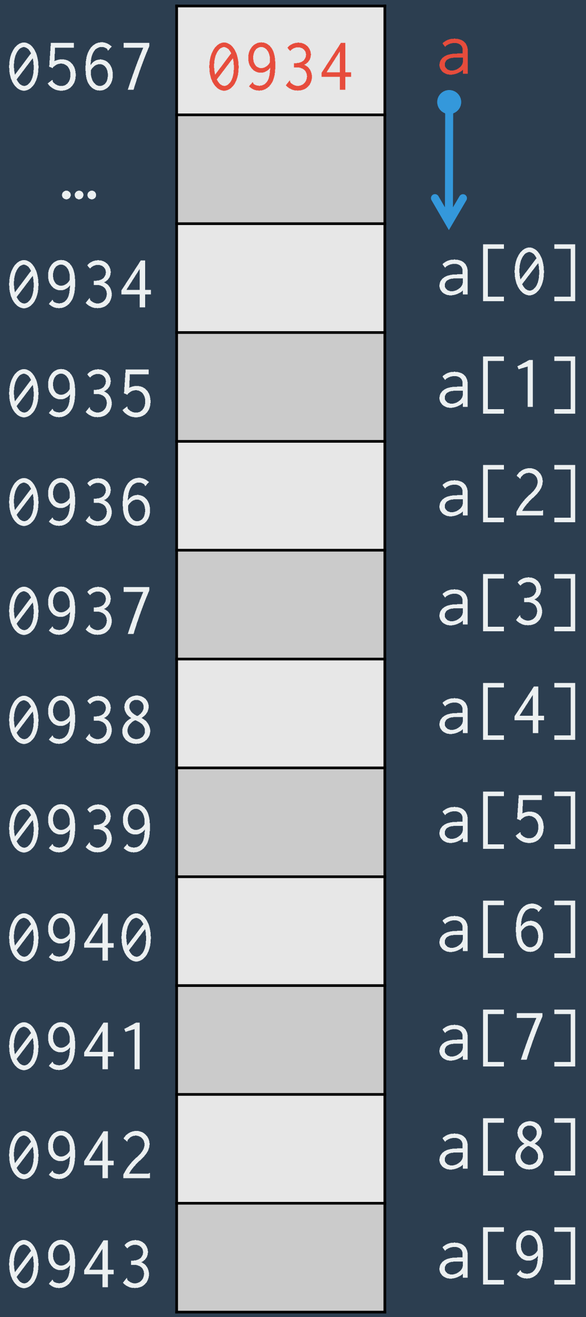
POINTER ARITHMETIC

!

The variable name is a (constant) pointer to the first element of the array.

A pointer can be used to access array elements.

Pointer arithmetic is done via the indirection operator (*).



0567 0934 a

…

0934 a[0]

0935 a[1]

0936 a[2]

0937 a[3]

0938 a[4]

0939 a[5]

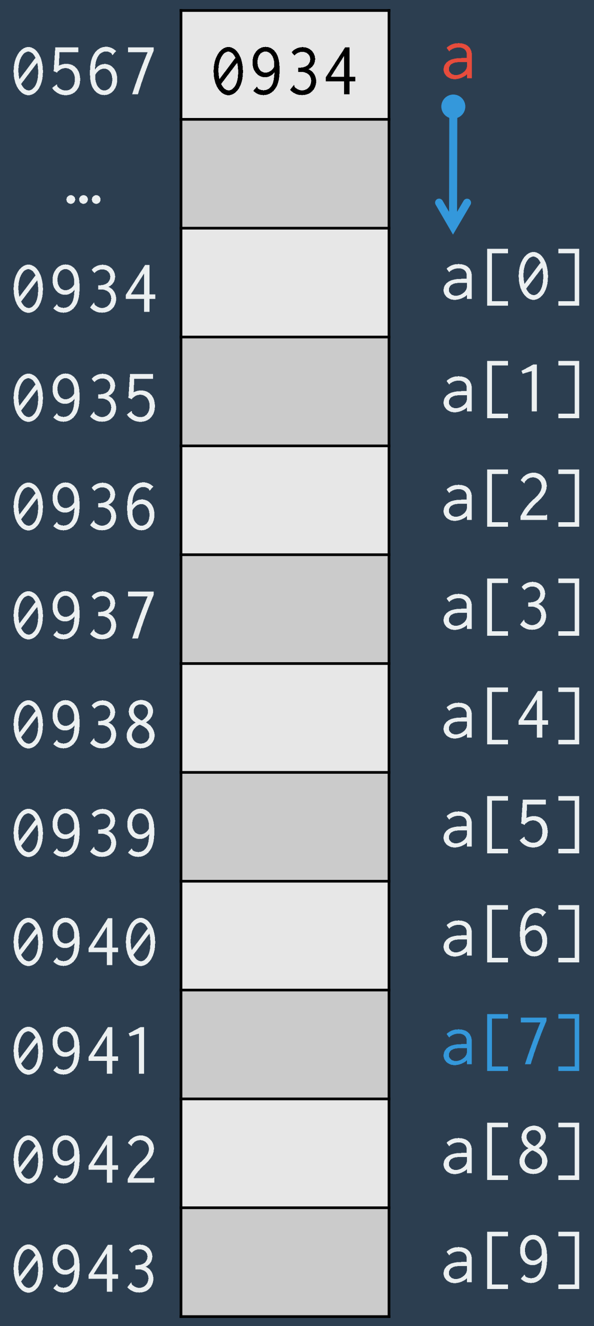
0940 a[6]

0941 a[7]

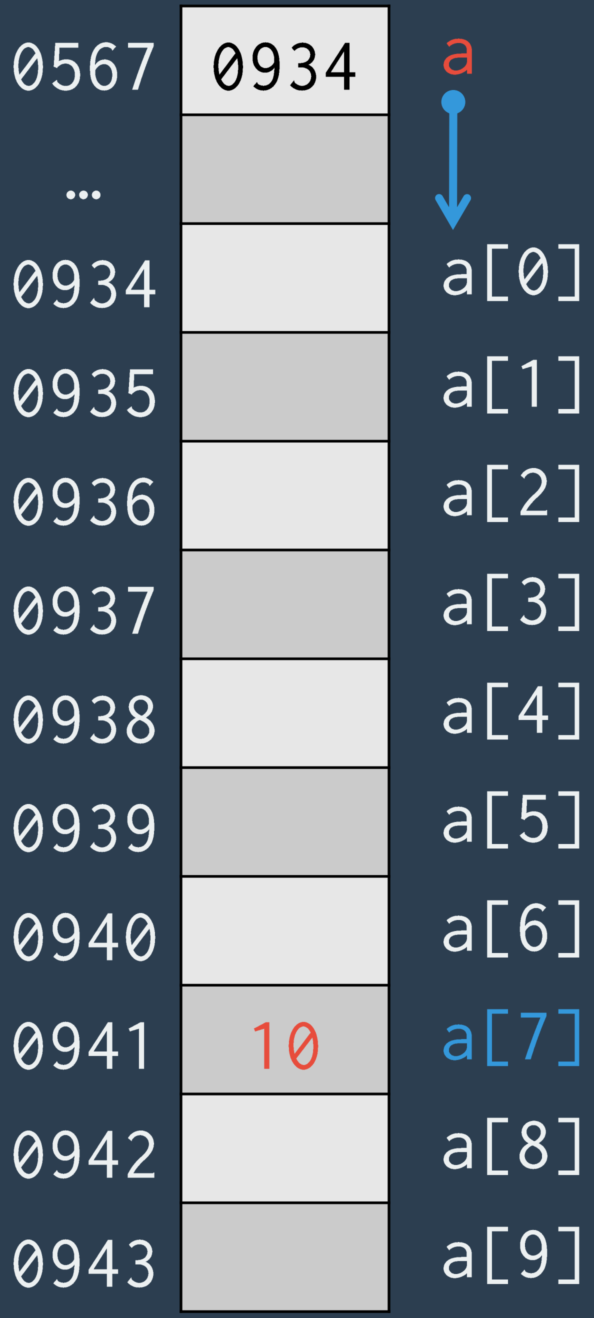
0942 a[8]

0943 a[9]

```
int a[10];
```



a[7] is equivalent to *(a + 7) or the 8th array element



`a[0]`

`*a` or `*(a+0)`

```
a[1]
```

```
*(a+1)
```

```
a[2]
```

```
*(a+2)
```

```
a[n-1]

        *(a+(n-1))
```

PROBLEM 3.

```
a[0] = 30; //same as “*a = 30”

scanf(“%d”, &a[9]); //same as (1)
printf(“%d”, a[2]); //same as (2)
scanf(“%d”, &a[0]); //same as (3)
```

Did you know that, normally, you can’t create a folder with the name “con” on any Microsoft OS?

Problem 3.

```
a[0] = 30; //same as “*a = 30”

scanf(“%d”, &a[9]); //same as a+9
printf(“%d”, a[2]); //same as *(a+2)
scanf(“%d”, &a[0]); //same as a
```

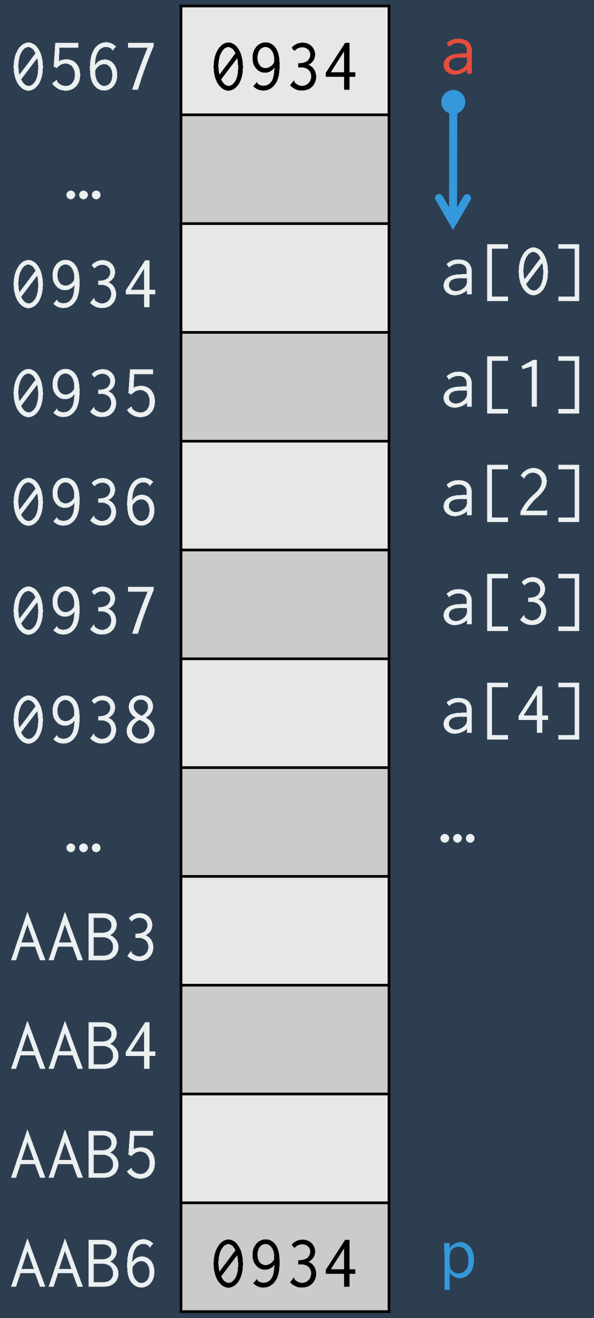
```
//arrays and loops
for(i=0; i<10; i++)
{
   scanf(“%d”, &a[i]);
   //scanf(“%d”, a+i);
}
```

Some notes

```
int a[5], *p;
p = a;

//Are these valid?
*(p+2) = 24;
p[8] = 7;
```

Pointers other than the array variable name can be used to access the array elements.



0567	0934	a
…		
0934		a[0]
0935		a[1]
0936		a[2]
0937		a[3]
0938		a[4]
…		…
AAB3		
AAB4		
AAB5		
AAB6	0934	p

p can be used to access the elements of a

The address operator (&) can be used to obtain the address of the ith element.

```
int a[10], *p;

p = &a[3];
//p now holds the address of
the 4th element
```

The variable name of an array cannot hold memory locations other than the array’s first element.

```
int a[10], b[20];
int x=8, *p;

p = &x;
//The ff. are invalid:
a = b;
b = p;
a = &x;
```

Arrays as parameters

To pass arrays as actual parameters to functions,…

…pass the array name without an index.

```
int main()
{
  int a[10];
  getInput(a);


}
```

```
int main()
{
  int a[10];
  getInput(a);
  /*the address of the first
    element is passed */
}
```

Arrays as formal parameters can be declared as:

```
//a pointer
int foo(int *p)
{

}
```

```
//an array w/ specified size
int foo(int p[10])
{

}
```

```
//or w/o specified size
int foo(int p[])
{

}
```