Pointers & Arrays

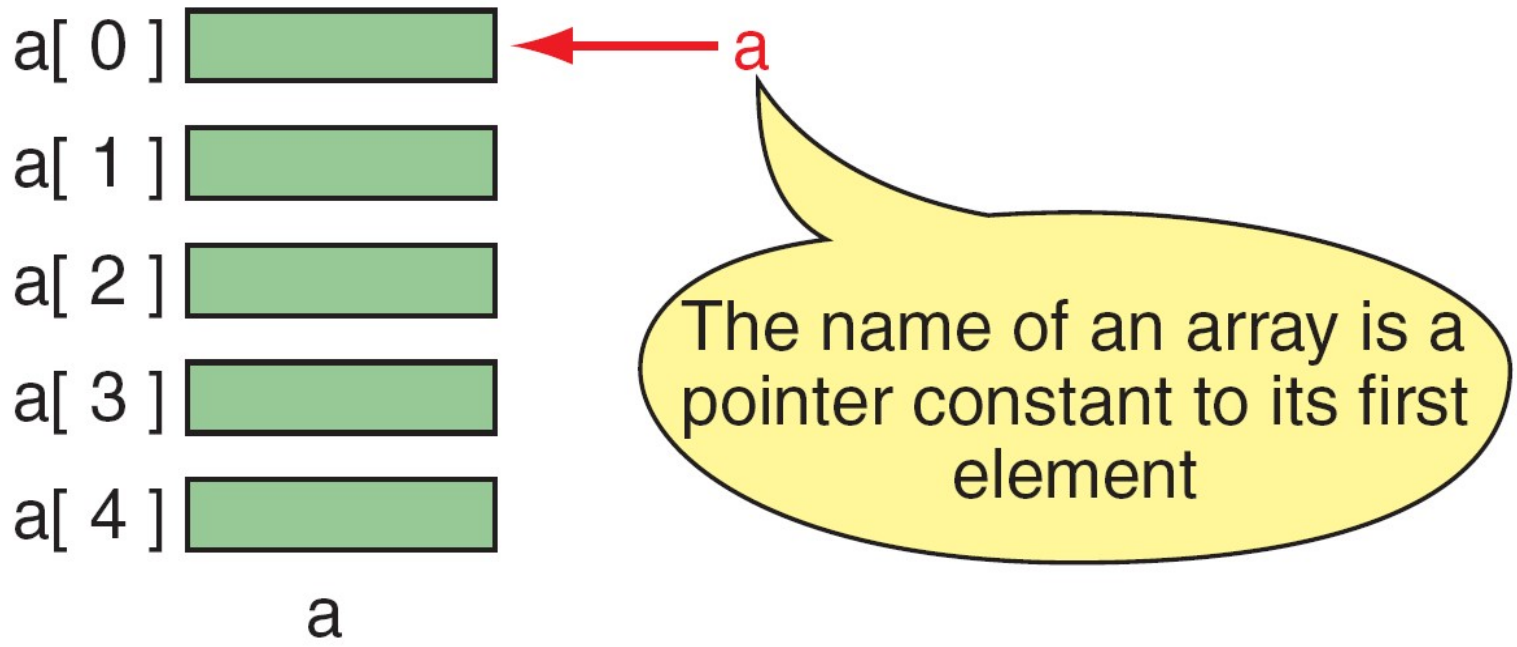
- Array variables are pointers to the array start
- Example:

```
char *ptr;
char str[10];
ptr = str; //ptr now points to array start
ptr = &str[0]; //Same as above line
```

- Array indexing is same as dereferencing after pointer addition.
- Example:

```
str[1] = 'a';
*(ptr+1) = 'a';// same as above line
```

Pointers & Arrays



Arrays and Pointers

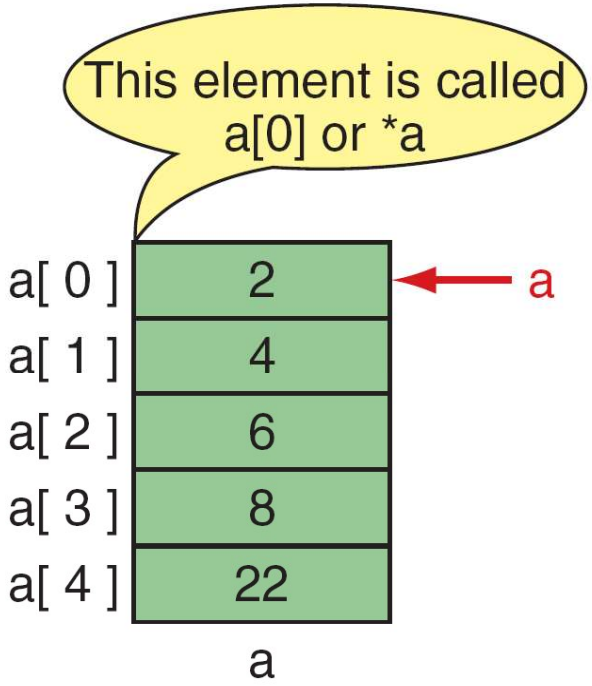
Note

same

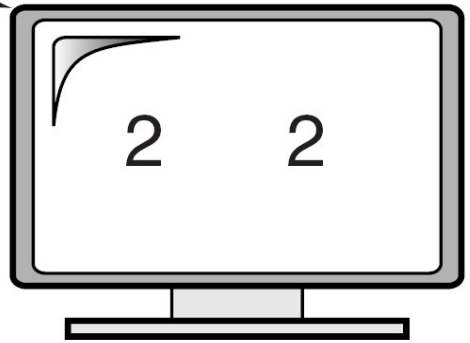
a ⟷ &a[0]

a is a pointer only to the first element—not the whole array.

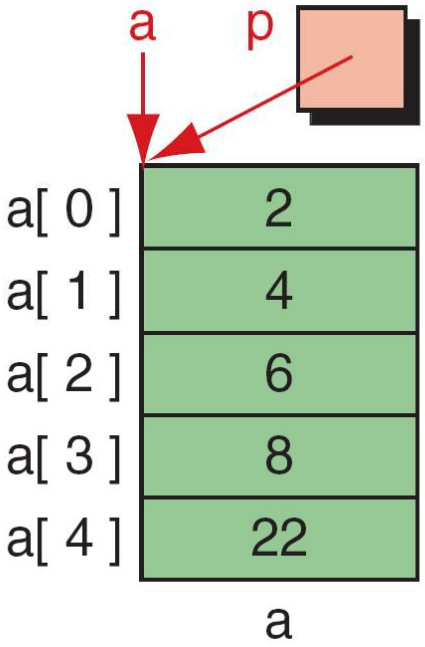
Arrays and Pointers



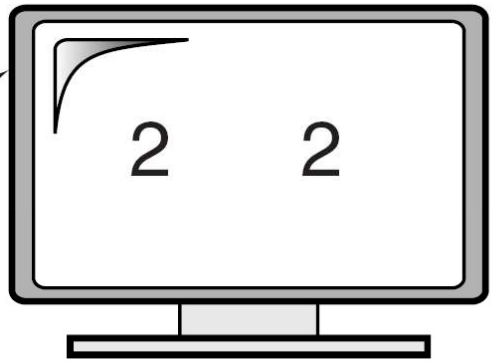

```c
#include <stdio.h>
int main (void)
{
   int a[5] = {2,4,6,8,22};
   printf("%d %d", *a, a[0]);

   return 0;
} // main
```



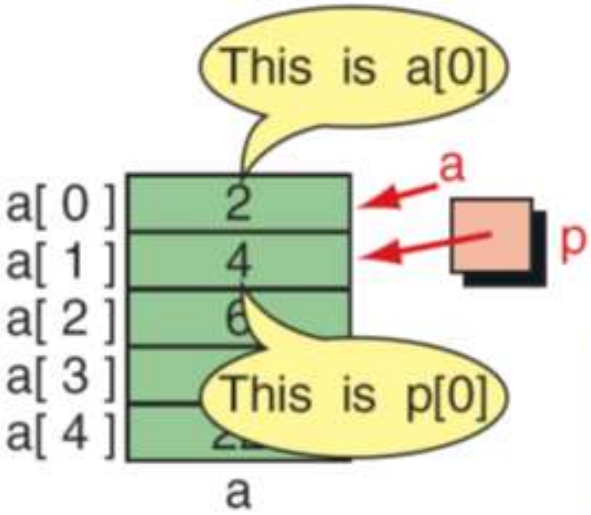
Arrays and Pointers



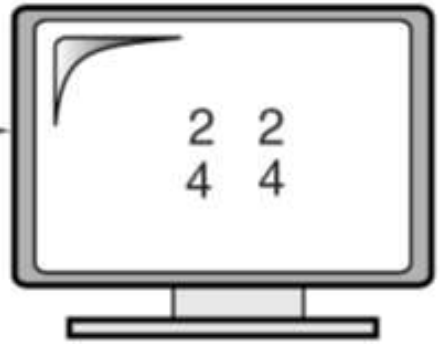

```
#include <stdio.h>
int main (void)
{
int  a[5] = {2, 4, 6, 8, 22};
int* p    =  a;
  …
  printf("%d %d\n", a[0], *p);
  …
  return 0;
} // main
```



Arrays and Pointers




```
#include <stdio.h>
int main (void)
{
  int  a[5] = {2, 4, 6, 8, 22};
  int* p;
  …
  p = &a[1];
  printf("%d %d", a[0], p[-1]);
  printf("\n");
  printf("%d %d", a[1], p[0]);
  …
} // main
```



Note

To access an array, any pointer to the first element can be used instead of the name of the array.

Pointer Expressions (Recap)

operation	Description
p++, p--	Increment (decrement) p to point to the next element, it is equivalent to p+=1 (p -=1)
p+i , p-i	Point to i-th element beyond (in front of) p but value of p is fixed
p[i]	Equivalent to p + i if p points to the start of array
p + n (integer)	n must be an integer, its meaning is offset
p - q	Offset between pointer p and pointer q
p+q, p*q, p/q, p%q	invalid
Relational operator of two pointers p, q	valid, including p > q, p < q, p == q, p >= q, p <= q, p != q

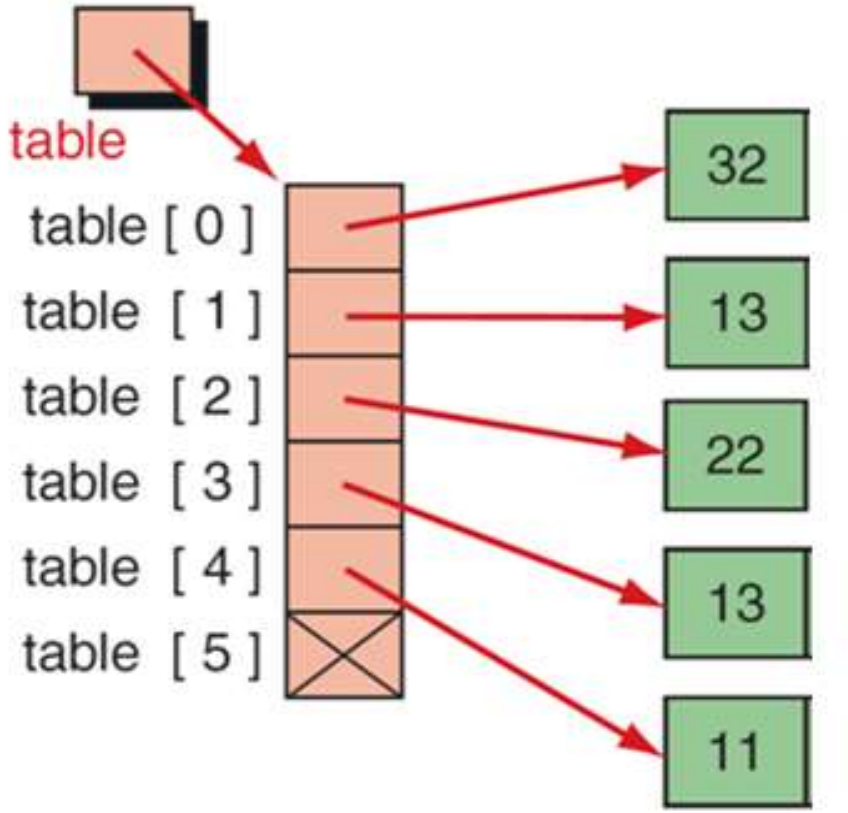
Pointers & Arrays-Example

```
#include <stdio.h>
int main(void)
 {    int i;
      int my_array[] = {1,23,17,4,-5,100}; //Compiler figures out size
      int *ptr;

      ptr = &my_array[0]; /* point our pointer to the first element of the
      array */
      printf("\n\n");
      for (i = 0; i < 6; i++)
            {
            printf("my_array[%d] = %d ",i,my_array[i]);
            printf("ptr + %d = %d\n",i, *(ptr + i));
            }
      return 0;
 }
```

```
my_array[0] = 1 ptr + 0 = 1
my_array[1] = 23 ptr + 1 = 23
my_array[2] = 17 ptr + 2 = 17
my_array[3] = 4 ptr + 3 = 4
my_array[4] = -5 ptr + 4 = -5
my_array[5] = 100 ptr + 5 = 100
```

Array of Pointers

- Arrays can contain pointers to …..



Array of Pointers

```
int *z[4] = {NULL, NULL, NULL, NULL};
int  a[4] = {1, 2, 3, 4};

z[0] = a;           // same as &a[0];
z[1] = a + 1;       // same as &a[1];
z[2] = a + 2;// same as &a[2];
z[3] = a + 3;// same as &a[3];

for (int x=0;x<4;x++)
 printf("\n %d --- %d ",a[x],*z[x]);
```

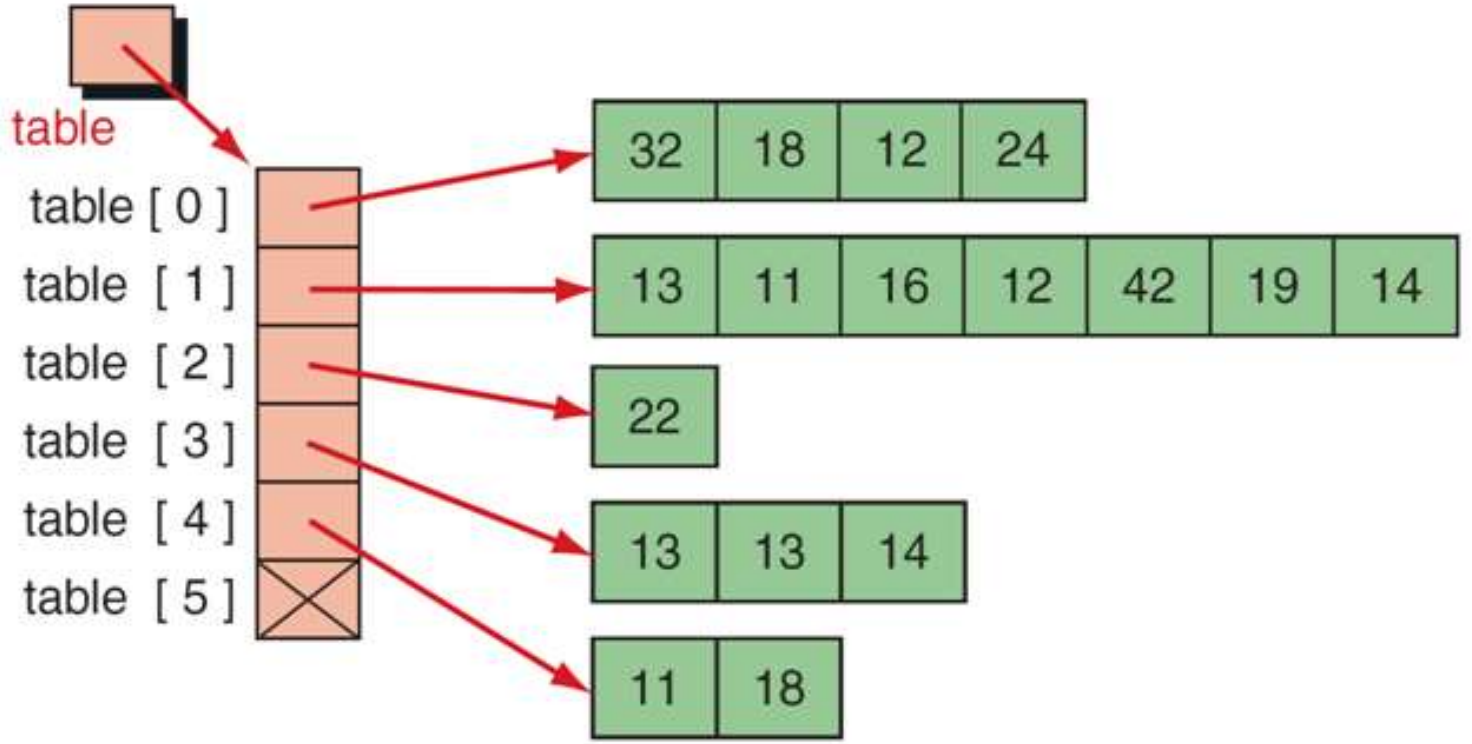
```
1 --- 1
2 --- 2
3 --- 3
4 --- 4
```

```
for (int x=0;x<4;x++)
 printf("\n %d --- %d ",a[x], z[x]);
```

```
1 --- 61877072
2 --- 61877076
3 --- 61877080
4 --- 61877084
```

Array of Pointers

- Arrays can contain pointers to (array)



Array of Pointers

```
int  x[] = {32, 18, 12, 24};
int  y[] = {13, 11, 16, 12, 42, 19, 14};
int *z[2] = {NULL, NULL};
int i;

z[0] = x;              // same as &x[0];
z[1] = y;             // same as  &y[0];

for (i=0;i<2;i++)
 printf("\n %d ",*z[i]);
```

```
32
13
```