Array

Index

01. Array

- Definition
- One-dimensional
- Multi-dimensional
- Initialization

Array

Definition - Scalar vs Aggregate Variables

Scalar is capable of holding a single data item

C also supports aggregate variables that can store collections of values

Two kinds of aggregates in C

- Array
- Structure

Array

One-dimensional Array

An array is a data structure containing a number of data values with same type

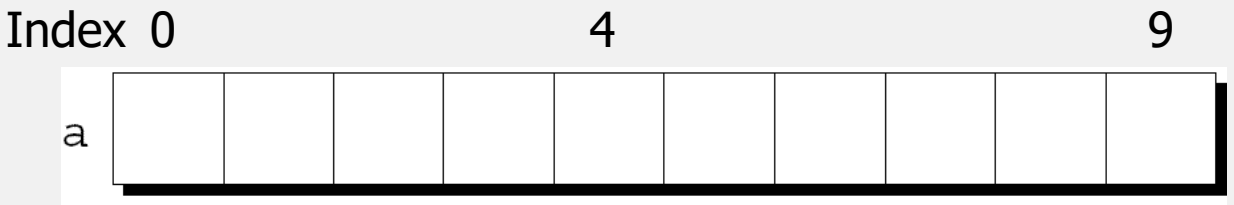
The values, i.e. elements, can be selected at their index individually

The elements of a one-dimensional array a are conceptually arranged one after another in a single row

How to declare an array?

```
int a[10];

#define N 10
int a[N];
```



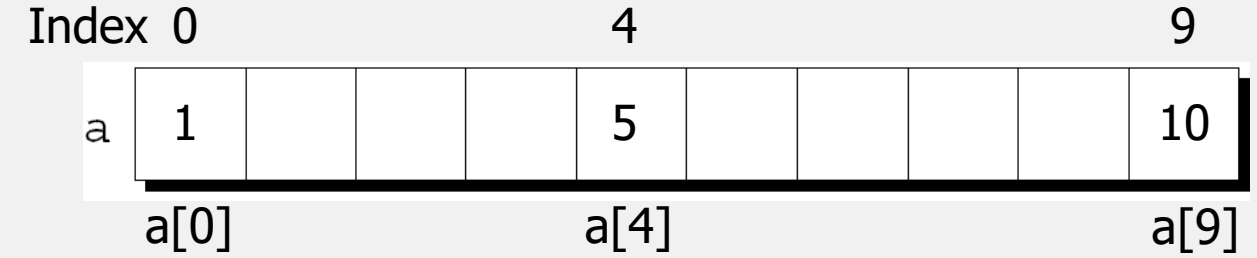
Array

One-dimensional Array

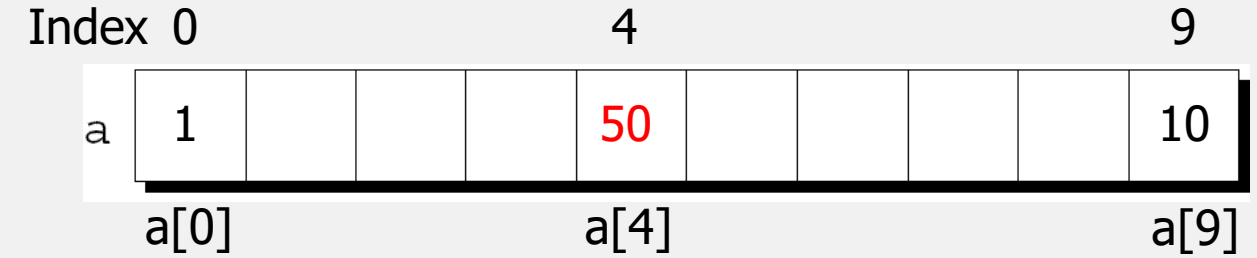
How to access elements in the array?

- Write the array name followed by an integer value in square bracket
  - This is referred to as subscripting or indexing the array
  - Important concept: the range of index is from 0 to N-1

```
int x, a[10];
x = a[0];      //x = 1
```




```
a[4] = 50;
```



- In general, if an array contains elements of type T, then each element of the array is treated as if it were a variable of type T

Array

One-dimensional Array

Loop expression is the array best friend

```
#define N 10
int a[N];
```

```
//The array initialization
for (int i = 0; i < N; i++)
{
    a[i] = 0;
}
```

```
//The array initialization
for (int i = 0; i < N; i++)
{
    scanf('%d", &a[i]);
}
```

```
//The array initialization
for (int i = 0; i < N;)
{
    scanf('%d", &a[i++]);
}
```

Array

One-dimensional Array

C doesn't require that subscript bounds be checked; if a subscript goes out of range, the program's behavior is undefined

A common mistake: forgetting that an array with n elements is indexed from 0 to n-1, not 1 to n

```
int a[10], i;

for (i = 1; i <= 10; i++)
    a[i] = 0;
```

Array

One-dimensional Array

An array subscript may be any integer expression

```
a[i+j*10] = 0;
```

The expression can even have side effects

```
i = 0;
while (i < N)
    a[i++] = 0;
```

Array

One-dimensional Array

Be careful when an array subscript has a side effect

```
i = 0;
while (i < N)
    a[i] = b[i++];
```

The expression a[i] = b[i++] accesses the value of i and also modifies i, causing undefined behavior

The problem can be avoided by removing the increment from the subscript

```
for (i = 0; i < N; i++)
    a[i] = b[i];
```

Array

One-dimensional Array

Write a program to reverse a series of entered numbers using array

```
Enter 10 numbers: 55 20 18 33 56 48 10 8 75 34
In reverse oder: 34 75 8 10 48 56 33 18 20 55
```

Array

Initialization

Write a program to present the repeated digits

```
Enter a number: 939577
Repeated digit(s): 7 9
```

```
Enter a number: 9527
No repeated digit
```

Array

Initialization

An array, like any other variable, can be given an initial value at the time it's declared

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int a[10] = {1, 2, 3, 4}; //initial value is {1, 2, 3, 4, 0, 0, 0, 0, 0, 0}

int a[10] = {0};          //initial value is {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

int a[10] = {6}; -> ?     //initial value is {6, 0, 0, 0, 0, 0, 0, 0, 0, 0}

int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Array

Initialization

It's often the case that relatively few elements of an array need to be initialized explicitly; the other elements can be given default values

Initial value is {0, 2, 0, 0, 0, 6, 0, 0, 0, 55}

```
int a[10] = {0, 2, 0, 0, 0, 6, 0, 0, 0, 55};
```

```
int a[10] = {[1] = 2, [5] = 6, [9] = 55};
```

```
int a[10] = {1, 2, 3, [5] = 6, 7, [8] = 11};
```

```
int a[] = {[1] = 2, [5] = 6, [9] = 55, [23] = 24};
```

What is the length of array a?

Array

Sizeof Operator

The sizeof operator can determine the size of an array (in bytes)

If a is an array of 10 integers, then sizeof(a) is typically 40 (assuming that each integer requires 4 bytes)

Hence, it can be also used to calculate the length of an array

```
int a[] = {[1] = 2, [5] = 6, [9] = 55, [23] = 24};
printf("The length of arrar a is: %d", (int)(sizeof(a)/sizeof(a[0])));
```

Some programmers use this expression when the length of the array is needed

A loop that clears the array a

```
for (i = 0; i < sizeof(a) / sizeof(a[0]); i++)
    a[i] = 0;
```

Array

Sizeof Operator

Some compliers produce a warning message for the expression

```
i < sizeof(a) / sizeof(a[0])
```

The variable i probably has type int (a signed type), whereas sizeof produces a value of type size_t (an unsigned type)

Comparing a signed integer with an unsigned integer can be dangerous, but in this case it's safe

How to avoid a warning?

```
for (i = 0; i < (int) (sizeof(a) / sizeof(a[0])); i++)
    a[i] = 0;
```

```
#define SIZE ((int) (sizeof(a) / sizeof(a[0])))

for (i = 0; i < SIZE; i++)
    a[i] = 0;
```

Array

Multi-dimensional Array

An array may have any number of dimensions

The following declaration creates a two-dimensional array (a matrix, in mathematical terminology)

int x[3][4];

	col 1	col 2	col 3	col 4
row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

To access the element of m in row i, column j, we must write m[i][j]

Don't use m[i, j] because C treats the comma as an operator in this context, so m[i, j] = m[j]

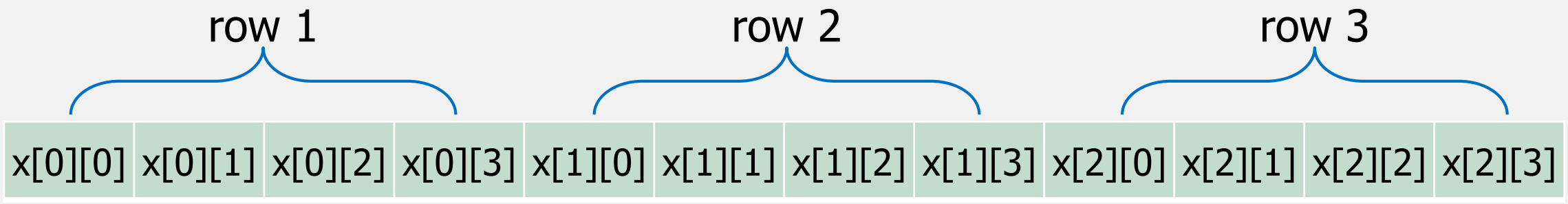
Array

Multi-dimensional Array

Although we visualize two-dimensional arrays as table, that's not the way they're actually stored in computer memory

int x[3][4];

	col 1	col 2	col 3	col 4
row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]



Array

Initialization

We can create an initializer for a two-dimensional array by nesting one-dimensional initializers

```
int x[3][4] = {{1, 1, 1, 1},
               {0, 1, 0, 1}};

int x[3][4] = {{1, 1, 1},
               {0, 0, 1, 0},
               {1, 0}};

double x[2][2] = {[0][0] = 1.0, [1][1] = 1.0};
```

Array

An Example

Write a program to deal a random hand of cards with constant arrays

```
Enter number of cards in hand: 6
Your hand: ah ks 3s 10c 3c qh
```

Additional C library and function for completing program

- time (from <time.h>) - returns the current time
  - time(NULL)
- srand (from <stdlib.h>) - initializes C's random number generator
  - srand((unsigned) time(NULL))
- rand (from <stdlib.h>) - produces an apparently random number
  - rand()