

CSE105

ARRAY (PART 1)

Scalar Variables versus Aggregate Variables

- So far, the only variables we've seen are ***scalar***: capable of holding a single data item.
- C also supports ***aggregate*** variables, which can store collections of values.
- There are two kinds of aggregates in C: ***arrays*** and ***structures***.

One-Dimensional Arrays

- An **array** is a data structure containing a number of data values, all of which have the same type.
- These values, known as **elements**, can be individually selected by their position within the array.
- The simplest kind of array has just one dimension.
- The elements of a one-dimensional array *a* are conceptually arranged one after another in a single row (or column):



Defining Arrays

- When defining arrays, specify
 - Name
 - Type of array (Array can be defined over **any type**)
 - Number of elements
`arrayType arrayName[numberOfElements];`
 - Examples:
`float grade[7];`
`int c[10];`
- Defining multiple arrays of same type
 - Format similar to regular variables
 - Example:
`int b[100], x[27];`

Array Elements

Name of array (Note that all elements of this array have the same name, c)

- To refer to an element, specify
 - **Array name** for the collection
 - **Position number** for the member
- **Format:**
 - arrayname* [*position number*]
 - First element at position 0
 - n element array named c:
 - ✦ c[0], c[1]...c[n - 1]

| | |
|-------|------|
| c[0] | -45 |
| c[1] | 6 |
| c[2] | 0 |
| c[3] | 72 |
| c[4] | 1543 |
| c[5] | -89 |
| c[6] | 0 |
| c[7] | 62 |
| c[8] | -3 |
| c[9] | 1 |
| c[10] | 6453 |
| c[11] | 78 |

Position number of the element within array c

One-Dimensional Arrays

- To declare an array, we must specify the *type* of the array's elements and the *number* of elements:

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

- The elements may be of **any type**; the length of the array can be any (integer) constant expression.
- Using a macro to define the length of an array is an excellent practice:

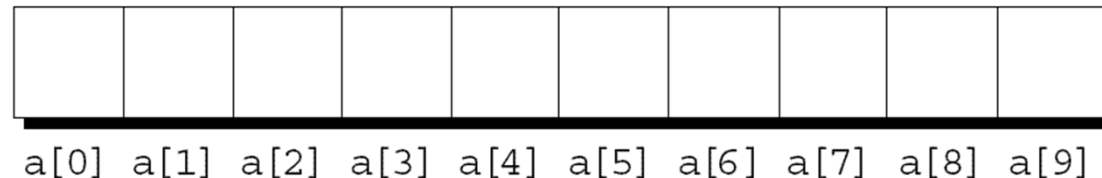
```
#define N 10
```

```
...
```

```
int a[N];
```

Array Subscripting

- To access an array element, write the array name followed by an integer value in square brackets.
- This is referred to as ***subscripting*** or ***indexing*** the array.
- The elements of an array of length n are indexed from 0 to $n - 1$.
- If a is an array of length 10, its elements are designated by $a[0]$, $a[1]$, ..., $a[9]$:



Array Subscripting

- Expressions of the form `a[i]` are lvalues, so they can be used in the same way as ordinary variables:

```
a[ 0 ] = 1;  
printf( "%d\n", a[ 5 ] );  
++a[ i ];
```

- 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 Subscripting

- Many programs contain `for` loops whose job is to perform some operation on every element in an array.
- Examples of typical operations on an array `a` of length `N`:

```
for (i = 0; i < N; i++)  
    a[i] = 0;                /* clears a */
```

```
for (i = 0; i < N; i++)  
    scanf("%d", &a[i]);      /* reads data into a */
```

```
for (i = 0; i < N; i++)  
    sum += a[i];             /* sums the elements of a */
```

Array Subscripting

- 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;
```

With some compilers, this innocent-looking `for` statement causes an infinite loop.

Array Subscripting

- 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;
```

Using Arrays: Averaging Grades

```
float grade[7], total=0;
/* initialize grades somehow */
total = grade[0] + grade[1] + grade[2] + grade[3] +
        grade[4] + grade[5] + grade[6];
printf("average = %f\n", total/7.0);
```

Or,

```
For (int i=0; i<7; i++)
    total+= grade[i];
printf("average = %f\n", total/7.0);
```

Actually, we can use **arrays** efficiently with **loops**

Program: Reversing a Series of Numbers

- The `reverse.c` program prompts the user to enter a series of numbers, then writes the numbers in reverse order:

Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31

In reverse order: 31 50 11 23 94 7 102 49 82 34

- The program stores the numbers in an array as they're read, then goes through the array backwards, printing the elements one by one.

reverse.c

```
/* Reverses a series of numbers */

#include <stdio.h>

int main(void)
{
    int a[10], i, N=10;

    printf("Enter %d numbers: ", N);
    for (i = 0; i < N; i++)
        scanf("%d", &a[i]);

    printf("In reverse order:");
    for (i = N - 1; i >= 0; i--)
        printf(" %d", a[i]);
    printf("\n");

    return 0;
}
```

Array Initialization

- An array, like any other variable, can be given an initial value at the time it's declared.
- The most common form of ***array initializer*** is a list of constant expressions enclosed in braces and separated by commas:

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

Array Initialization

- If the initializer is shorter than the array, the remaining elements of the array are given the value 0:

```
int a[10] = {1, 2, 3, 4, 5, 6};  
/* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

- Using this feature, we can easily initialize an array to all zeros:

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

There's a single 0 inside the braces because it's illegal for an initializer to be completely empty.

- It's also illegal for an initializer to be longer than the array it initializes.

Array Initialization

- If an initializer is present, the length of the array may be omitted:

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

- The compiler uses the length of the initializer to determine how long the array is.

Static Array Sizes

- The size of array should be **static**; no dynamic array in C
 - The following is wrong:

```
int x;  
scanf("%d", &x);  
int temp[x];
```

You should use dynamic memory allocation like **malloc()** for dynamically-sized arrays

C Does Not Check Array Bounds

```
double grade[7];  
int i = 9;  
grade[i] = 3.5; /* Is i out-of-range? If so, any error message ?*/
```

- You should check it for yourself if not sure

```
if (0 <= i && i < 7)  
    /* OK to access grade[i] */  
else  
    printf("Array Index %d out-of-range.\n", i);
```

Program: Checking a Number for Repeated Digits

- The `repdigit.c` program checks whether any of the digits in a number appear more than once.
- After the user enters a number, the program prints either `Repeated digit` or `No repeated digit`:

```
Enter a number: 28212  
Repeated digit
```

- The number 28212 has a repeated digit (2); a number like 9357 doesn't.

Program: Checking a Number for Repeated Digits

- The program uses an array of 10 Integer values to keep track of which digits appear in a number.
- Initially, every element of the `digit_seen` array is 0.
- When given a number `n`, the program examines `n`'s digits one at a time, storing the current digit in a variable named `digit`.
 - If `digit_seen[digit]` is `>1`, then `digit` appears at least twice in `n`.
 - If `digit_seen[digit]` is 0, then `digit` has not been seen before.

repdigit.c

```
/* Checks numbers for repeated digits */

#include <stdio.h>

int main(void)
{
    int digit_seen[10] = {0};
    int digit;
    long n;

    printf("Enter a number: ");
    scanf("%ld", &n);
    while (n > 0) {
        digit = n % 10;
        digit_seen[digit]++;
        n /= 10;
    }
}
```

Try This

- Create an integer array of 20 elements.
- Initialize the array with 0.
- Insert 5 numbers in an array from keyboard
- Detect whether the array is sorted in descending/ascending order.
- Insert a number at the end of the array
- Insert a number at the beginning of the array
- Delete the first number from the array.
- Delete lowest element from the array.

Try This

- Search a number from the array.
- Find the highest and lowest number from array (with their position)
- Factorial of $n = n * (n-1) * \dots * 3 * 2 * 1$
- Fibonacci sequence $n = 0 \ 1 \ 1 \ 2 \ 3 \ 5 \ \dots \ \dots \ \dots \ n^{\text{th}} \text{ term}$
- Minimal Element Sort