One-Dimensional Array

• Generic declaration:
  
  `typedef variablename[size];`

• Example: `int a[10];`
  
  -- Defines an array of `int` with subscripts ranging from 0 to 9
  
  -- There are `10*sizeof(int)` bytes of memory reserved for this array.


  -- You can use `a[0]=10; x=a[2]; a[3]=a[2];` etc.
  
  -- You can use `*(a)=10; x=* (a+2); *(a+3)=*(a+2);` etc.

Array-Bounds Checking

• C DOES NOT check array bounds index during:
  
  -- Compilation
  
  -- Runtime

• If you access an array using index that is outside the bound:
  
  -- may get “something...”
  
  -- may get a memory exception (segmentation fault, bus error, core dump error)

• It is the programmer’s responsibility to ensure that the programs are correctly written and debugged!

Initializing 1D Arrays

• Example:

  ```
  int array[4] = {100, 200, 300, 400};
  ```

  -- This is equivalent to:

  ```
  int array[4] = {100, 200, 300, 400};
  ```

  -- You can also let the compiler figure out the array size:

  ```
  int array[] = {100, 200, 300, 400};
  ```
Two-Dimensional Array

• In C, a two-dimensional array is stored as a one-dimensional array, where each of the elements is a one-dimensional array.
• `int a[4][3]; int *p = &a[0][0];`

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
<td>a2</td>
</tr>
<tr>
<td>a3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- `a` is a constant pointer points to `a[0][0]`.
- `p` is a pointer variable points to `a[0][0]`.

2D Array vs. Pointers Array

int `a[4][3];`
• `a` is a 2-d array with 12 int-sized memory allocated.
int `*b[4];`
• `b` is a 1-d array with 4 int-pointer-sized memory allocated.

- Each element of `b` can points to a 1-D array with different length.
- The pointer values in `b` have to be initialized before they can be used.
  ```
  b[0]=a[0];
  b[1]=a[1];
  b[2]=a[2];
  b[3]=a[3];
  ```

2-D Array Operations

```c
int a[2][3]={{1,10,100},
            {4,40,400}};
int x=**a; // x=7
x=*a[0]; // x=7
// change row
x=(*a+1)[0]; // x=7
x=*(*(a+1)); // x=7
x=*a[1]; // x=7
// change column
x=(*a+1); // x=7
x=*(*(a)+1); // x=7
x=*(a[0]+1); // x=7
```

2-D Array Function Parameter Declaration

- A 2-D array function parameter declaration has to include the number of columns; the number of rows is irrelevant.
- Calling `fun1`:
  ```
  int a[4][3]={{1,2,3}, {4,5,6}, {7,8,9},
              {10,11,12}};
  fun1(a);
  ```
```
2D Array Names as Function Arguments

• Similar to 1D array, passing a 2D array name as a function argument is equivalent to passing a pointer variable to the function.

  //both function declarations are equivalent
  fun1(int a[]){…}
  fun1(int* a){…}

• The array name becomes a pointer variable within the function.