Array

Topics

- What is an array?
- Declaration and initialization
- Search as an example

Arrays

- An array is a collection of data elements that are of the same type (e.g., a collection of integers,characters, doubles)
- The dimension/size of an array must be known at programming time, and cannot be changed during program execution
- Arrays are like flats in a building, or post office boxes
- Arrays provide a good way to name a collection, and to reference its individual elements.

Array Applications

- Given an array of test scores, determine the maximum and minimum scores.
- Read in an array of student names and rearrange them in alphabetical order (sorting).
- Given the height measurements of students in a class, output the names of those students who are taller than average.

Array Declaration

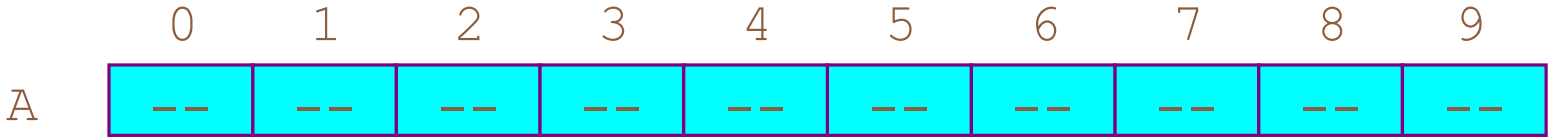
- Syntax:

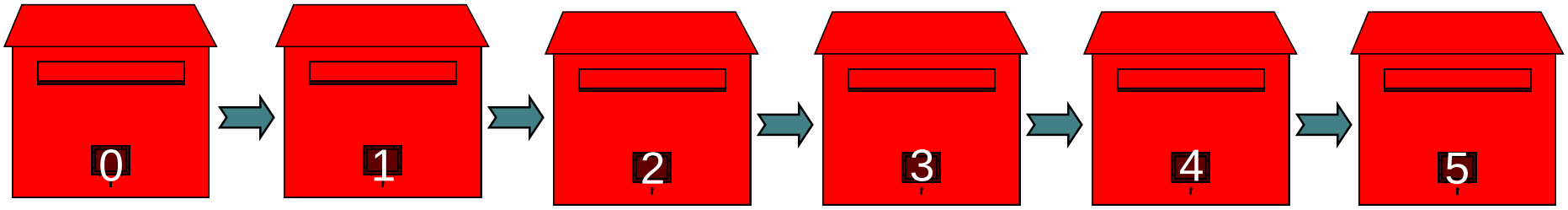
  `<type> <arrayName>[<dimension>]`

- The array elements are all values of the type `<type>`
- The size of the array is indicated by `<dimension>`, the number of elements in the array
- `<dimension>` must be an `int` constant or a constant expression. Note that it is possible for an array to have multiple dimensions.

Array Declaration Example

```
// array of 10 uninitialized ints
int A[10];
```





Subscripting

- **Suppose**

```
int A[10];    // array of 10 ints
```

- An array occupies a contiguous block of memory in computer. The content of the memory is not defined at declaration.
- To access an individual element we must apply a subscript to array name `A`
  - A subscript is a bracketed expression
    - The expression in the brackets is known as the index
  - First element of array has index 0

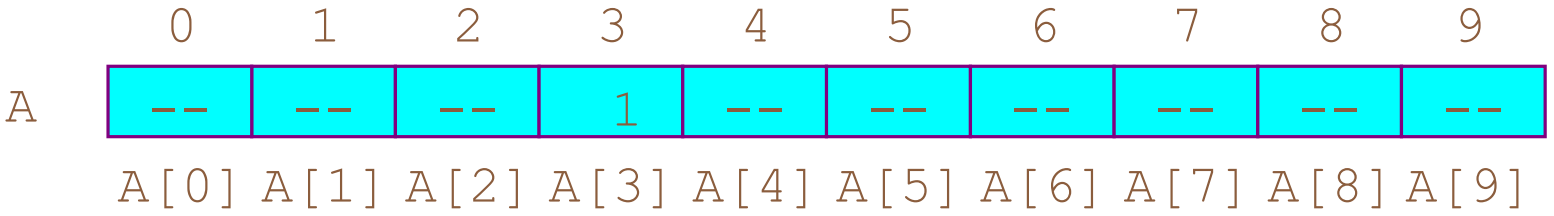
    `A[0]`
  - Second element of arry has index 1, and so on

    `A[1]`
  - Last element has an index one less than the size of the array

    `A[9]`
- Incorrect indexing is a common error

Subscripting

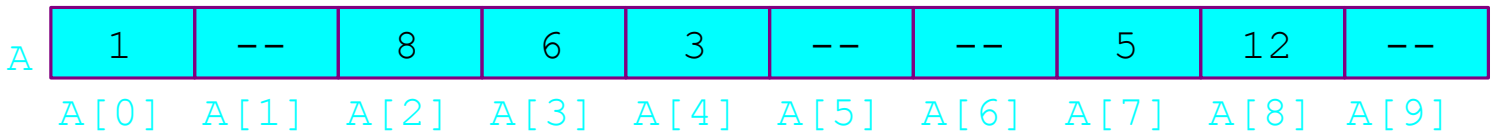
```
// array of 10 uninitialized ints
int A[10];

A[3] = 1;
int x = A[3];
```



Array Element Manipulation

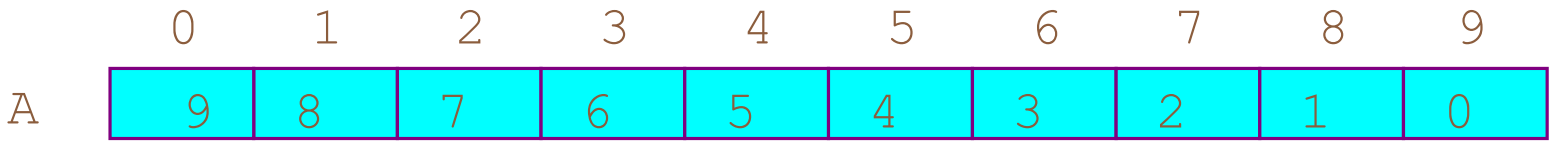
- Consider

```
int A[10], i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
cin >> A[k]; // where the next input value is 3
```

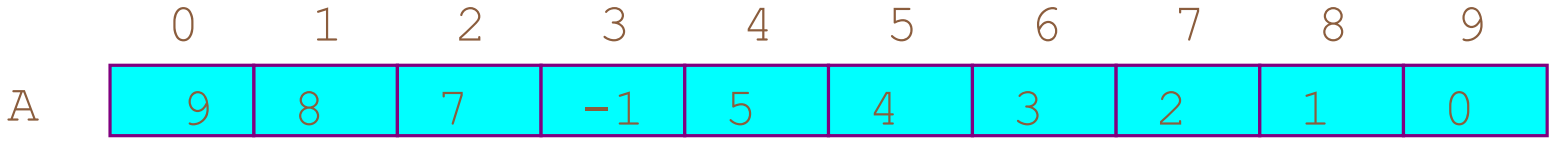


Array Initialization

```
int A[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
```




```
A[3] = -1;
```



Example Definitions

- Suppose

```
const int N = 20;
const int M = 40;
const int MaxStringSize = 80;
const int MaxListSize = 1000;
int P = 10;  //array size has to be known at compile time
```

- Then the followings are all legal array definitions.

```
float x[ P ];              // array of 10 floats
int A[10];                 // array of 10 ints
char B[MaxStringSize];     // array of 80 chars
double C[M*N];             // array of 800 doubles
int Values[MaxListSize];// array of 1000 ints
```

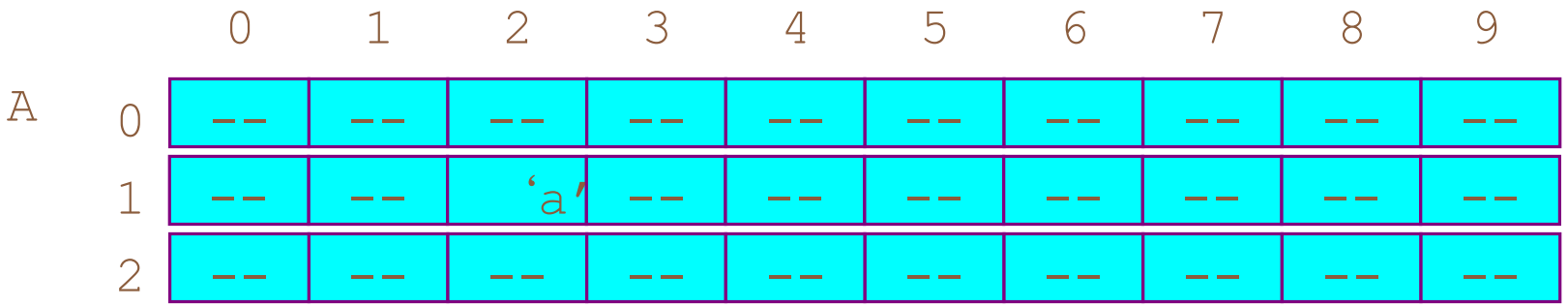
2-D Array Example

```
// 2-D array of 30 uninitialized ints
int A[3][10];
```

A	0	1	2	3	4	5	6	7	8	9
0	--	--	--	--	--	--	--	--	--	--
1	--	--	--	--	--	--	--	--	--	--
2	--	--	--	--	--	--	--	--	--	--

2-D Array References

```
// 2-D array of 30 uninitialized chars
char A[3][10];

A[1][2] = 'a';
char resp = A[1][2];
```



Array Access

- An array, no matter what dimension it is, is represented internally as a block of linearly increasing memory address
  - The compiler will keep track of the mapping between indexes and addresses
  - The name of the array refers to the starting address of the array
- `int A[i]` is accessed in memory address `A + i*4`
- For 2-D array, each row is of size `4 * MAX_COL`
- `int A[i][j]` for `A[i][MAX_COL]` is accessed in memory location

  ```
  A + i * 4 * MAX_COL + 4 * j
  ```
- `int A[i][j][k]` for `A[i][MAX_1][MAX_2]` is accessed in memory address

  ```
  A + i * 4 * (MAX_1 * MAX_2) +
      j * 4 * MAX_2 + k * 4
  ```

Inputting an Array

```
const int MaxSize = 10000;
int A[MaxSize];
int n = 0;
int CurrentInput;
while(n< MaxSize &&  cin>>CurrentInput){
      A[n] = CurrentInput;
      n++;
}
```

Evaluate this first;
if this fails, no need to evaluate the
next condition

Fails if user hits ^D

```
A[n++] =
CurrentInput;
```

Displaying an Array

```
// Array A of n elements has already been set
int i;
for (i=0; i<n; i++)
     cout << A[i] << " ";
cout << endl;
```

Smallest Value

- Problem
  - Find the smallest value in an array of integers
- Input
  - An array of integers and a value indicating the number of integers
- Output
  - Smallest value in the array
- Note
  - Array remains unchanged after finding the smallest value!

Smallest Value: Preliminary Design

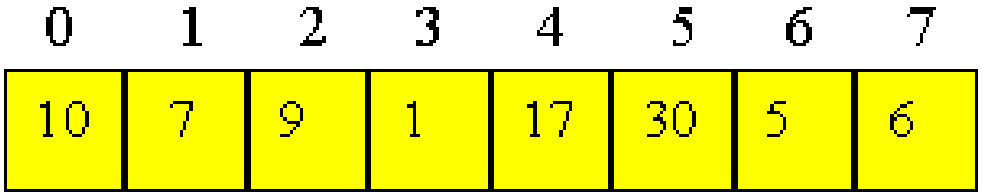
- Idea
  - When looking for smallest value, need a way of remembering best candidate found so far
- Design
  - Search array looking for smallest value
    - Use a loop to consider each element in turn
    - If current element is smallest so far, then update smallest value so far
  - When done examining all of the elements, the smallest value seen so far is the smallest value

Smallest Value

```
const int N=10;
int A[N];
int SmallestValueSoFar, i;

...   // A[] is input by user

SmallestValueSoFar = A[0];
for (i=1; i<N; i++)
    if (A[i] < SmallestValueSoFar)
          SmallestValueSoFar = A[i];
```

Unordered Linear Search

- Search an unordered array of integers for a value and save its index if the value is found. Otherwise, set index to -1.



- Algorithm:

```
Assume value not in array and set index = -1
Start with the first array element (index 0)
while(more elements in array){
      If value found at current index, save index
      Try next element (increment index)
}
```

Unordered Linear Search

```
void main() {

  int A[] = {10, 7, 9, 1, 17, 30, 5, 6};
  int x, n, index;

  cout << "Enter search element: ";
  cin >> x;
  index = -1;
  for(n=0; n<8; n++)
       if(A[n]==x)
         index = n;
  if(index==-1)
       cout << "Not found!!" << endl;
  else
       cout << "Found at: " << index << endl;
}
```

returns the largest index which has the value; may run the index down and break out the loop once a match is found

```
for(n=7; n<=0; n--)
  if(A[n]==x){
    index = n;
    break;
  }
```

char array

```
char h[] = "hello";  // A null terminated string:
                     // char h[] = {'h', 'e', 'l', 'l', 'o', '\0'}
cout << sizeof( h ) << endl;   // get output 6

char b[] = "hello\0abcde";     // middle null character
cout << sizeof( b ) << endl;   // get output 12, with invisible trailing \0
cout <<  b  << endl;           // print out hello only (no abcde)
cout << b + 2 << endl;         // print out llo

char * p = "hello";           // allocation on stack
```

2-D `char` array

```
char str[][6] = {"hello", "there", "!"}; // the index must be
                                         // 6 or larger; otherwise compiler complains
                                         // allocate a 2-D char array
                                         // allocation on stack
cout << str[1] << endl;                    // print out there
cout << str[1][2] << end;                  // print out e

char *a[] = {"hello", "there", "!"};  // declaring an array of pointers
                                      // pointing to strings
                                      // allocation on stack
cout << a[1] << endl;                 // print out there
cout << a[1][3] << endl;              // print out r
```