Arrays

Mechanism for representing lists

Lists

- Problem solving often requires information be viewed as a list
  - List may be one-dimensional or multidimensional
- C++ provides two list mechanisms
  - Arrays
    - Traditional and important because of legacy libraries
    - Restrictions on its use
  - Container classes
    - First-class list representation
    - Common containers provided by STL
      - Vector, queue, stack, map, …
    - Preferred long-term programming practice

Array Terminology

- List is composed of *elements*
- Elements in a list have a *common name*
  - The list as a whole is referenced through the common name
- List elements are of the same type — the *base* type
- Elements of a list are referenced by *subscripting* or *indexing* the common name

C++ Restrictions

- Subscripts are denoted as expressions within brackets: [ ]
- Base type can be any fundamental, library-defined, or programmer--defined type
- The index type is integer and the index range must be 0 … **`n`**-1
  - where **`n`** is a programmer-defined constant expression.
- Parameter passing style
  - Always call by reference (no indication necessary)

Basic Array Definition

```
BaseType Id [ SizeExp ] ;
```

Type of values in list

Name of list

Bracketed constant expression indicating number of elements in list

```
double X [ 100 ] ;

            // Subscripts are 0 through 99
```

Example Definitions

- Suppose

```
const int N = 20;
const int M = 40;
const int MaxStringSize = 80;
const int MaxListSize = 1000;
```

- Then the following are all correct array definitions

```
int A[10];                 // array of 10 ints
char B[MaxStringSize];     // array of 80 chars
double C[M*N];             // array of 800 floats
int Values[MaxListSize];   // array of 1000 ints
Rational D[N-15];          // array of 5 Rationals
```

Subscripting

- Suppose

```
int A[10];    // array of 10 ints A[0], … A[9]
```

- To access individual element must apply a subscript to list name **A**
  - A subscript is a bracketed expression also known as the index
  - First element of list has index 0

    ```
    A[0]
    ```

  - Second element of list has index 1, and so on

    ```
    A[1]
    ```

  - Last element has an index one less than the size of the list

    ```
    A[9]
    ```

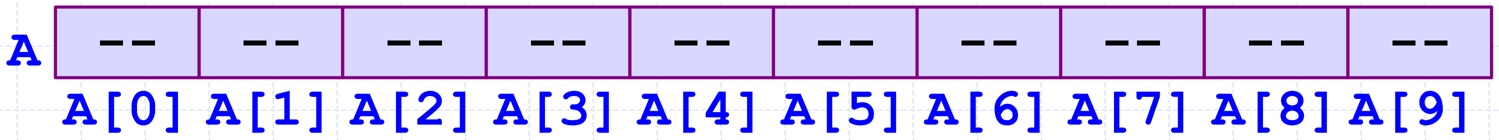
  - Incorrect indexing is a common error

    ```
    A[10]     // does not exist
    ```

Array Elements

- Suppose

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

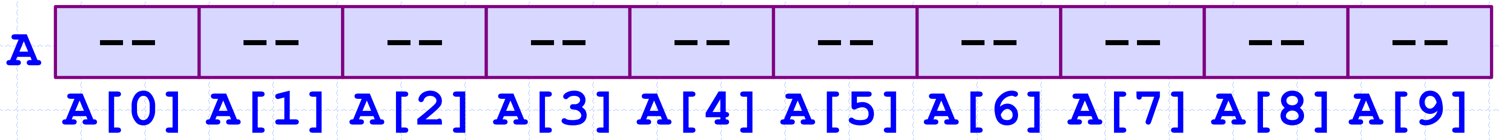


- To access an individual element we must apply a subscript to list name `A`

Array Element Manipulation

- Consider

```
int 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 next input value is 3
```



Array Element Manipulation

- Consider

```
int 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 next input value is 3
```

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

Array Element Manipulation

- Consider

```
int 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 next input value is 3
```

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

Array Element Manipulation

- Consider

```
int 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 next input value is 3
```

A	1	--	8	6	3	--	--	5	12	--
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Extracting Values For A List

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

Displaying A List

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

Smallest Value

- Problem
  - Find the smallest value in a list of integers
- Input
  - A list of integers and a value indicating the number of integers
- Output
  - Smallest value in the list
- Note
  - List remains unchanged after finding the smallest value!

Passing An Array

Notice brackets are empty

```
int ListMinimum(const int A[], int asize) {
   assert(asize >= 1);
   int SmallestValueSoFar = A[0];
   for (int i = 1; i < asize; ++i) {
      if (A[i] < SmallestValueSoFar ) {
         SmallestValueSoFar = A[i];
       }
   }
   return SmallestValueSoFar ;
}
```

Could we just assign a 0 and have it work?

Using ListMinimum()

- What happens with the following?

```
int Number[6];
Number[0] = 3; Number[1] = 88; Number[2] = -7;
Number[3] = 9; Number[4] = 1;  Number[5] = 24;

cout << ListMinimum(Number, 6) << endl;
```

Notice no brackets

```
int List[3];
List[0] = 9;   List[1] = 12;   List[2] = 45;

cout << ListMinimum(List, 3) << endl;
```

Remember

- Arrays are always passed by reference
  - Artifact of C
- Can use `const` if array elements are not to be modified
- Do not need to include the array size when defining an array parameter

Some Useful Functions

```
void DisplayList(const int A[], int n) {
   for (int i = 0; i < n; ++i) {
      cout << A[i] << " ";
   }
   cout << endl;
}
void GetList(int A[], int &n, int MaxN = 100) {
  for (n = 0; (n < MaxN) && (cin >> A[n]); ++n) {
      continue;
   }
}
```

Useful Functions Being Used

```
const int MaxNumberValues = 25;
int Values[MaxNumberValues];
int NumberValues;

GetList(Values, NumberValues, MaxNumberValues);
DisplayList(Values, NumberValues);
```

Multi-Dimensional Arrays

- Syntax

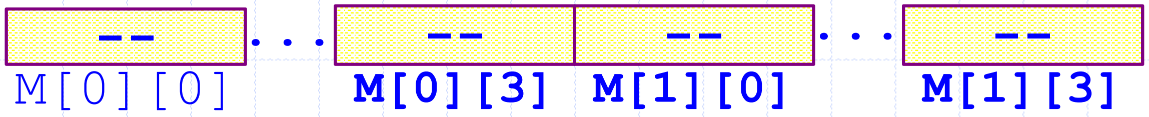
  `btype mdarray[size_1][size_2] ... [size_k]`
- Where
  - k - dimensional array
  - **`mdarray`**: array identifier
  - **`size_i`**: a positive constant expression
  - **`btype`**: standard type or a previously defined user type and is the base type of the array elements
- Semantics
  - **`mdarray`** is an object whose elements are indexed by a sequence of **`k`** subscripts
  - the **`i`**-th subscript is in the range **`0 ... size_i - 1`**

Memory Layout

- Multidimensional arrays are laid out in row-major order
- Consider

  `int M[2][4];`

- `M` is two-dimensional array that consists of 2 subarrays each with 4 elements.
  - 2 rows of 4 elements
- The array is assigned to a contiguous section of memory
  - The first row occupies the first portion
  - The second row occupies the second portion



Identity Matrix Initialization

```
const int MaxSize = 25;
float A[MaxSize][MaxSize];
int nr = PromptAndRead();
int nc = PromptAndRead();
assert((nr <= MaxSize) && (nc <= MaxSize));
for (int r = 0; r < nr; ++r) {
   for (int c = 0; c < nc; ++c) {
      A[r][c] = 0;
   }
   A[r][r] = 1;
}
```

Matrix Addition Solution

Notice only first brackets are empty

```
void MatrixAdd(const float A[][MaxCols],
 const float B[][MaxCols], float C[][MaxCols],
 int m, int n) {
   for (int r = 0; r < m; ++r {
      for (int c = 0; c < n; ++c) {
         C[r][c] = A[r][c] + B[r][c];
      }
   }
}
```