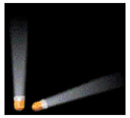
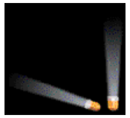
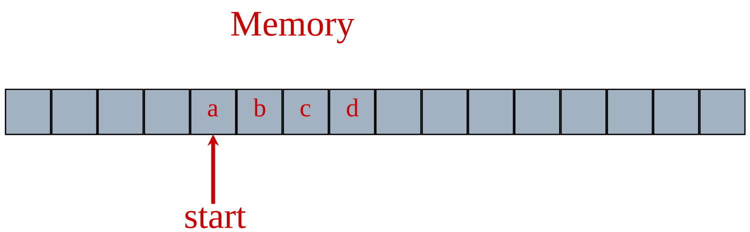
Representation of Arrays



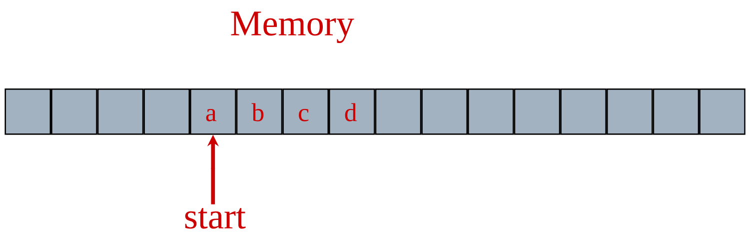


1D Array Representation In C++



- 1-dimensional array x = [a, b, c, d]
- map into contiguous memory locations
  - location(x[i]) = start + i

Space Overhead



space overhead = 4 bytes for start (memory address)

(excludes space needed for the elements of x)

2D Arrays

The elements of a 2-dimensional array a declared as:

int [][]a = new int[3][4];

may be shown as a table

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Rows Of A 2D Array

a[0][0] a[0][1] a[0][2] a[0][3] row 0

a[1][0] a[1][1] a[1][2] a[1][3] row 1

a[2][0] a[2][1] a[2][2] a[2][3] row 2

Columns Of A 2D Array

column 0	column 1	column 2	column 3
a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

2D Array Representation In C++

2-dimensional array x

a, b, c, d

e, f, g, h

i, j, k, l

view 2D array as a 1D array of rows

x = [row0, row1, row 2]

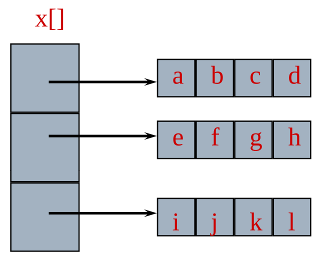
row 0 = [a,b, c, d]

row 1 = [e, f, g, h]

row 2 = [i, j, k, l]

and store as 4 1D arrays

Space Overhead



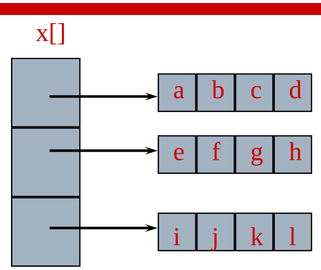
space overhead = overhead for 4 1D arrays

= 4 * 4 bytes

= 16 bytes

= (number of rows + 1) x 4 bytes

Array Representation In C++



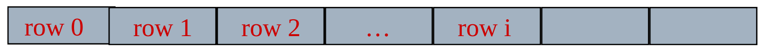
- This representation is called the array-of-arrays representation.
- Requires contiguous memory of size 3, 4, 4, and 4 for the 4 1D arrays.
- 1 memory block of size number of rows and number of rows blocks of size number of columns

Row-Major Mapping

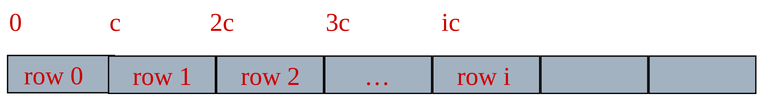
- Example 3 x 4 array:

a b c d
e f g h
i j k l

- Convert into 1D array y by collecting elements by rows.
- Within a row elements are collected from left to right.
- Rows are collected from top to bottom.
- We get y[] {a, b, c, d, e, f, g, h, i, j, k, l}



Locating Element x[i][j]



- assume x has r rows and c columns
- each row has c elements
- i rows to the left of row i
- so ic elements to the left of x[i][0]
- so x[i][j] is mapped to position ic + j of the 1D array

For n-dim Array

- For Array $a[u_1][u_2][u_3]..[u_n]$
- The position for $a[i_1][i_2][i_3]..[i_n]$

$$
\begin{aligned}
&= i_1 u_2 u_3 \ldots u_n + \\
&\quad i_2 u_3 u_4 \ldots u_n + \\
&\quad i_3 u_4 u_5 \ldots u_n + \\
&\quad \ldots \qquad + \\
&\quad i_{n-1} u_n + i_n
\end{aligned}
$$

Space Overhead for Row-major Mapping

4 bytes for start of 1D array +
4 bytes for c (number of columns)
= 8 bytes → Fixed! Doesn't change with array size

Compare to array of array representations:
(number of rows + 1) x 4 bytes

Remark: C++ use row-major mapping

Disadvantage

Row major mapping:
- Need contiguous memory of size rc.

Array of array representation:

Column-Major Mapping

a b c d
e f g h
i j k l

- Convert into 1D array y by collecting elements by columns.
- Within a column elements are collected from top to bottom.
- Columns are collected from left to right.
- We get y = {a, e, i, b, f, j, c, g, k, d, h, l}

In Class Exercise: Address of an element

- Assume A is an array and size of each element is 1. The address of A[3,3] is at 121 and A[6,4] is at 159. Find the address of the element A[4,5].(Hint: Consider different address mapping of array A.)

Matrix

Table of values. Has rows and columns, but numbering begins at 1 rather than 0.

a b c d	row 1
e f g h	row 2
i j k l	row 3

- Use notation x(i,j) rather than x[i][j].
- May use a 2D array to represent a matrix.

Shortcomings Of Using A 2D Array For A Matrix

- Indexes are off by 1.
- C++ arrays do not support matrix operations such as add, transpose, multiply, and so on.
  - Suppose that x and y are 2D arrays. Can't do x + y, x –y, x * y, etc.
- Develop a class Matrix for object-oriented support of all matrix operations.

Diagonal Matrix

1 0 0 0
0 2 0 0
0 0 3 0
0 0 0 4

Store in 1D array

1 2 3 4

- x(i,j) is on diagonal iff i = j
- number of diagonal elements in an n x n matrix is n
- non diagonal elements are zero
- store diagonal only vs n^2 whole

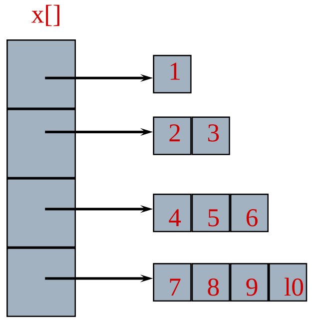
Lower Triangular Matrix (See Fig. 2.8 in P121)

An n x n matrix in which all nonzero terms are either on or below the diagonal.

1 0 0 0
2 3 0 0
4 5 6 0
7 8 9 10

- x(i,j) is part of lower triangle iff i >= j.
- number of elements in lower triangle is 1 + 2 + … + n = n(n+1)/2.
- store only the lower triangle

Array Of Arrays Representation



Use an irregular 2-D array … length of rows is not required to be the same.

Creating An Irregular Array

```
// declare a two-dimensional array variable
// and allocate the desired number of rows
int ** irregularArray = new int* [numberOfRows];

// now allocate space for the elements in each row
for (int i = 0; i < numberOfRows; i++)
  irregularArray[i] = new int [length[i]];
```

Map Lower Triangular Array Into A 1D Array

Use row-major order, but omit terms that are not part of the lower triangle.

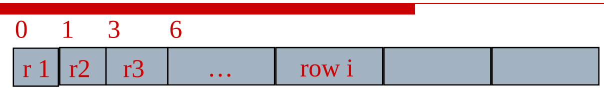
For the matrix

1	0	0	0
2	3	0	0
4	5	6	0
7	8	9	10

we get

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Index Of Element [i][j]



- Order is: row 1, row 2, row 3, …
- Row i is preceded by rows 1, 2, …, i-1
- Size of row i is i.
- Number of elements that precede row i is
  $1 + 2 + 3 + \ldots + i-1 = i(i-1)/2$
- So element (i,j) is at position $i(i-1)/2 + j - 1$ of the 1D array.