Data Structure and Algorithms

Deepali Londhe
Information Technology,
PICT, Pune

1

Agenda

2

- Array
- Single and multidimensional array address calculation
- Recursion.

2

Which mathematical concept refers to collection?

3

- **Sets**
- First we specify a common property among "things" and then we gather up all the "things" that have this common property.
- E.g. set of things you read, set of items you wear, set of things you eat etc.
  - {socks, shoes, watches, shirts, ...}
- Numerical Sets
  - Set of even numbers: {..., −4, −2, 0, 2, 4, ...}
  - Set of odd numbers: {..., −3, −1, 1, 3, ...}
  - Set of prime numbers: {2, 3, 5, 7, 11, 13, 17, ...}
  - Positive multiples of 3 that are less than 10: {3, 6, 9}
- When we define a set, all we have to specify is a common characteristic
- In mathematics, a set is a well-defined collection of distinct objects, considered as an object in its own right. The arrangement of the objects in the set does not matter. A set may be denoted by placing its objects between a pair of curly braces.

Array

4

14	12	3	445	511	50	.	.	.	.	.

Array

- An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.
- An array is collection of items stored at contiguous memory locations.
- An *array* is a finite, ordered and collection of homogeneous data elements
- **Finite:** because it contains only limited number of elements
- **Ordered:** as all the elements are stored one by one in contiguous locations of computer memory in a linear ordered fashion
- **Homogeneous:** all elements of an array are of the same data type only

Arrays

5

Array: a set of pairs (index and value)

data structure

For each index, there is a value associated with that index.

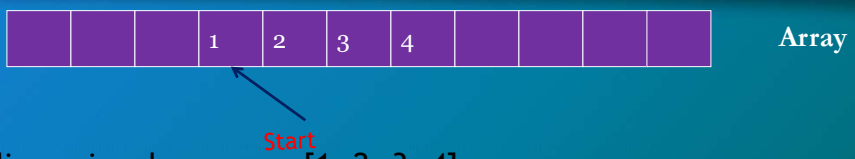
representation (possible)

implemented by using consecutive memory.

5

Array..

6



1-dimensional array x = [1, 2, 3, 4]
map into contiguous memory locations
location(x[i]) = start + i

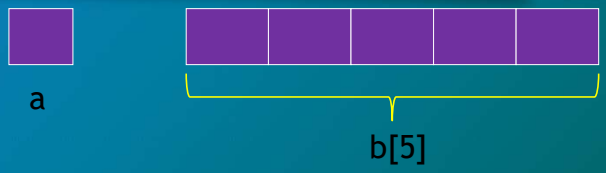
- An array is collection of items stored at contiguous memory locations. The idea is to store multiple items of same type together.
- This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).

6

Arrays in C++

7

```
int a=5;
int b[5];
int b[5]={5,3,2,1,6};
```



- Array declaration
  type name [elements];
- Initializing arrays
  - int b[5] = { }; //array of 5 ints each initialized to 0
  - int b[ ] = {5,3,2,1,6}; //size of array=number of values
- Accessing the values of an array
  - int b[2]=45;
  - X=b[3];

Other Operations on arrays
b[2] = a;
b[a] = 75;
x = foo [a+2];
b[b[a]] = b[2] + 5;

7

8

```
//============================================
// Name        : arrayPrint.cpp
// Author      : Deepali
// Version     : 1.1
// Copyright   : Your copyright notice
// Description : print elements of array
//============================================
#include <iostream>
using namespace std;
void print(int b[], int size)
{
cout << "The elements of array are ->";
for(int i=0;i<size;i++)
cout<<"  "<<b[i];
}

int main() {
int size;
int a[5]={3,4,5,6,7};
size=sizeof(a)/sizeof(int); //calculate sizeof array
print(a,size);
return 0;
}
```

The elements of array are -> 3 4 5 6 7

8

```
int main() {
    int size;
    int a[5]={3,4,5,6,7};
    size=sizeof(a)/sizeof(int); //calculate sizeof array
    print(a,size);
    elementAdd(a,size);
return 0;
}
```

9

```
void printAdress(int b[], int size)
{
cout<<"The address of elements are ->"<<endl;
for(int i=0;i<size;i++)
    cout<<"b["<<i<<"]--> "<<& b[i]<<endl;

}
```

```
The address of elements are ->
b[0]--> 0x6dfeec
b[1]--> 0x6dfef0
b[2]--> 0x6dfef4
b[3]--> 0x6dfef8
b[4]--> 0x6dfefc
```

3	4	5	6	7
7208684	7208688	7208692	7208696	7208700

location(x[i]) = start + i

9

Arrays in C++..

10

int list[5], *plist[5];

list[5]: five integers
list[0], list[1], list[2], list[3], list[4]
*plist[5]: five pointers to integers
plist[0], plist[1], plist[2], plist[3], plist[4]

implementation of 1-D array

list[0]	base address = α
list[1]	α + sizeof(int)
list[2]	α + 2*sizeof(int)
list[3]	α + 3*sizeof(int)
list[4]	α + 4*size(int)

Compare int *list1 and int list2[5] in C++.
Same: list1 and list2 are pointers.
Difference: list2 reserves five locations.

Notations:
list2 - a pointer to list2[0]
(list2 + i) - a pointer to list2[i] (&list2[i])
*(list2 + i) - list2[i]

10

11

```
#include <iostream>
using namespace std;
void printPtr(int *p, int size)
{
cout<<"The Base address is-->"<<int(p)<<endl;
cout<<"The elements and address of elements are ->"<<endl;
for(int i=0;i<size;i++)
   cout<<"p["<<i<<"]--> "<<"Value="<<*(p+i)<< "  Address is --
   >"<<int(p+i)<<endl;
}

int main() {
int size;
int a[5]={3,4,5,6,7};
size=sizeof(a)/sizeof(int);
printPtr(a,size);
return 0;
}
```

```
The Base address is-->7208684
The elements and address of elements are ->
p[0]--> Value=3  Address is -->7208684
p[1]--> Value=4  Address is -->7208688
p[2]--> Value=5  Address is -->7208692
p[3]--> Value=6  Address is -->7208696
p[4]--> Value=7  Address is -->7208700
```


11

Array 2-D

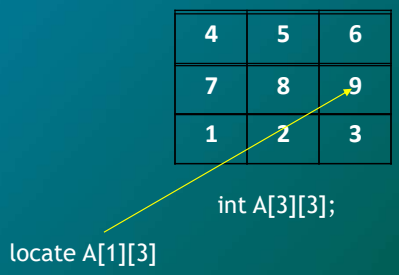
12

2D- a bidimensional array can be imagined as a two-dimensional table made of elements, all of them of a same uniform data type

The elements of a 2-dimensional array a declared as:
int a[3][4];

may be shown as a table

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
Column-0	Column-1	Column-2	Column-3	



12

2D Array Representation in C++

13

2-dimensional array A

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

view 2D array as a 1D array of rows

A = [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

13

14

```
void addrTD(int b[][3],int s)
{
for(int i=0;i<s;i++)
   for(int j=0;j<s;j++)
      cout<<"b["<<i<<"]["<<j<<"]"<<b[i][j]<<
         "  Address->"<<(int)&b[i][j]<<endl;
}
void TDarrayInput()
{
int A[3][3];
cout<<"Enter the elements of 3*3 array-->"
for(int i=0;i<3;i++)
   for(int j=0;j<3;j++)
      cin>> A[i][j];

addrTD(A,3);//get the address
}

int main() {
TDarrayInput();
return 0;
}
```

```
Enter the elements of 3*3 array-->
1 2 3
4 5 6
7 8 9
b[0][0]1  Address->7208588
b[0][1]2  Address->7208592
b[0][2]3  Address->7208596
b[1][0]4  Address->7208600
b[1][1]5  Address->7208604
b[1][2]6  Address->7208608
b[2][0]7  Address->7208612
b[2][1]8  Address->7208616
b[2][2]9  Address->7208620
```


14

The Array ADT

15

Objects: A set of pairs <index, value> where for each value of index there is a value from the set item. Index is a finite ordered set of one ormore dimensions, for example, {0, … , n-1} for one dimension, {(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)} for two dimensions, etc.

Methods:
for all A ∈ Array, i ∈ index, x ∈ item, j, size ∈ integer

Array Create(j, list) ::= **return** an array of j dimensions where list is a j-tuple whose kth element is the size of the kth dimension. Items are undefined.

Item Retrieve(A, i) ::= **if** (i ∈ index) **return** the item associated with index value i in array A
else return error

Array Store(A, i, x) ::= **if** (i in index)
return an array that is identical to array A except the new pair <i, x> has been inserted **else return** error

15

Questions

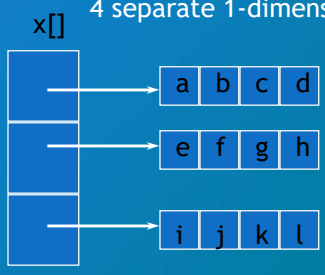
16

- What is the complexity of "retrieve" in an array?
- What is the complexity of "store" in an array?
- What about insertion and deletion for ordered elements in an arrary?

16

2D Array Representation in C++

17



space overhead = overhead for 4 1D arrays
= 4 * 4 bytes
= 16 bytes
= (number of rows + 1) x 4 bytes

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

17

Row major and Column Major

18

Row-major order

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Column-major order

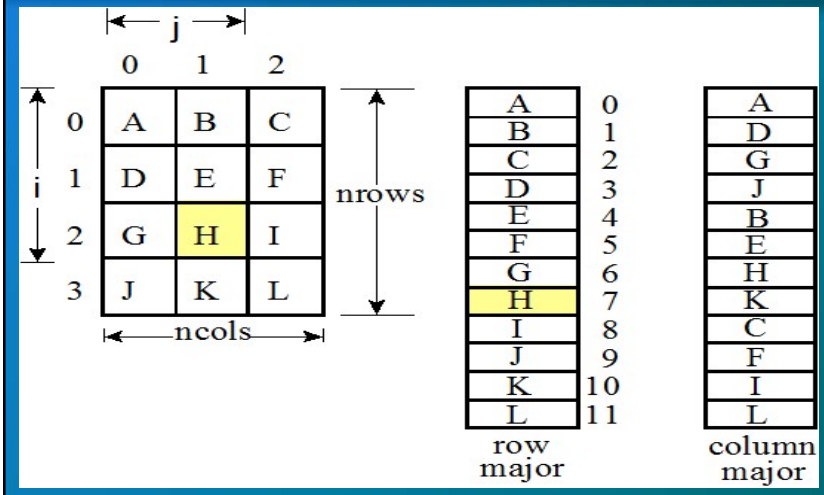
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

- In computing, row-major order and column-major order are methods for storing multidimensional arrays in linear storage such as random access memory.
- The difference between the orders lies in which elements of an array are contiguous in memory. In row-major order, the consecutive elements of a row reside next to each other, whereas the same holds true for consecutive elements of a column in column-major order.

18

Row major and Column Major

19



19

Row-Major Mapping

20

- Example 3 x 4 array:

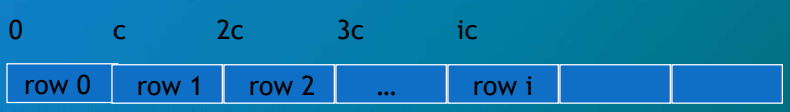
a b c d
e f g h
i j k l

- 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}

20

Locating Element x[i][j]

21



- 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]**
- x[i][j] is mapped to position
  ic + j of the 1D array

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

Locating x[1][2] i.e. f
1*4+2=6 position of 1D

Space Overhead

22

4 bytes for start of 1D array +4 bytes for c (number of columns)
= 8 bytes

Note that we need contiguous memory of size rc.

Column-Major Mapping

23

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**}

23

Row- and Column-Major Mappings

24

2D Array int a[3][6];

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]

0 1 2 3 4 5	0 3 6 9 12 15
6 7 8 9 10 11	1 4 7 10 13 16
12 13 14 15 16 17	2 5 8 11 14 17
(a) Row-major mapping	(b) Column-major mapping

24

Row- and Column-Major Mappings

25

- Row-major order mapping functions
  $map(i_1,i_2) = i_1u_2+i_2$ for 2D arrays
  $map(i_1,i_2,i_3) = i_1u_2u_3+i_2u_3+i_3$ for 3D arrays
- What is the mapping function for Figure 7.2(a)?
  $map(i_1,i_2) = 6i_1+i_2$
  $map(2,3) = ?$
- Column-major order mapping functions
  // do this as an exercise

Matrices

26

- *m x n* matrix is a table with *m* rows and *n* columns.
- *M(i,j)* denotes the element
- Common matrix operations
  - transpose
  - addition
  - multiplication

	col 1	col 2	col 3	col 4
row 1	7	2	0	9
row 2	0	1	0	5
row 3	6	4	2	0
row 4	8	2	7	3
row 5	1	4	9	6

Matrix Operations

27

- Transpose
  - The result of transposing an *m x n* matrix is an *n x m* matrix with property:

$$M^T(j,i) = M(i,j),\ 1 <= i <= m,\ 1 <= j <= n$$

- Addition
  - The sum of matrices is only defined for matrices that have the same dimensions.
  - The sum of two *m x n* matrices A and B is an *m x n* matrix with the property:

$$C(i,j) = A(i,j) + B(i,j),\ 1 <= i <= m,\ 1 <= j <= n$$

Matrix Operations

28

- Multiplication
  - The product of matrices A and B is only defined when the number of columns in A is equal to the number of rows in B.
  - Let A be *m x n* matrix and B be a *n x q* matrix. A*B will produce an *m x q* matrix with the following property:

$$C(i,j) = \Sigma(k=1..n)\ A(i,k) * B(k,j)$$

where $1 <= i <= m$ and $1 <= j <= q$

A Matrix Class

29

- There are many possible implementations for matrices.

```
// use a built-in 2 dimensional array
T matrix[m][n]
// use the Array2D class
Array2D<T> matrix(m,n)
// or flatten the matrix into a one-dimensional array
template<class T>
class Matrix {
        private:  int rows, columns;
                          T *data;
};
```


Shortcomings of using a 2D Array for a Matrix

30

- 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. Cannot do x + y, x -y, x * y, etc. in C++.
- We need to develop a class matrix for object-oriented support of all matrix operations.

31

31

References

32

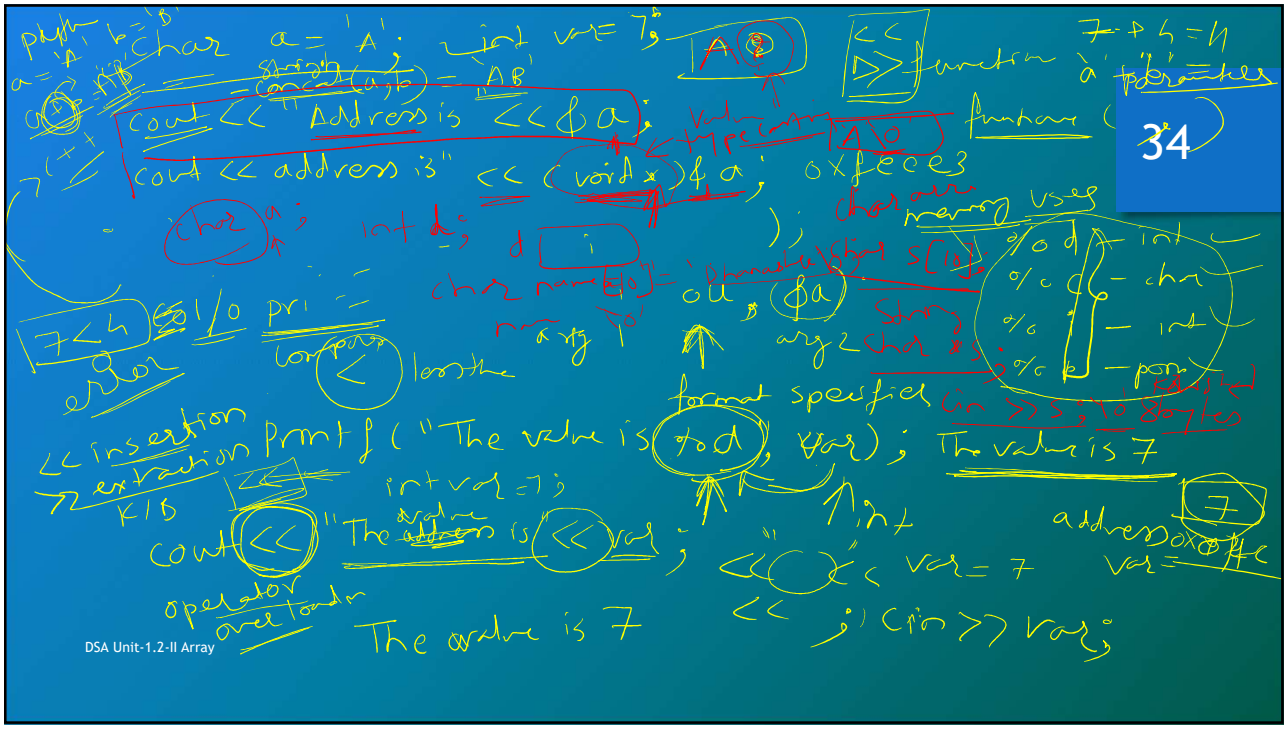
- **Books**
- Samanta Debasis, **"CLASSIC DATA STRUCTURES", PHI, 2nd ed.**
- Ellis Horowitz and Sartaj Sahni , "Fundamentals of Data Structures", Computer Science Press, 1983
- R. Gilberg, B. Forouzan, "Data Structures: A pseudo Code Approach with C++", Cengage Learning, ISBN 9788131503140.
- E. Horowitz, S. Sahni, D. Mehta, "Fundamentals of Data Structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN 16782928
- Dinesh P. Shah, Sartaj Sahani , "Handbook of DATA STRUCTURES and APPLICATIONS", CHAPMAN & HALL/CRC
- Bayer B. et al. (2015) Electro-Mechanical Brake Systems. In: Winner H., Hakuli S., Lotz F., Singer C. (eds) Handbook of Driver Assistance Systems. Springer, Cham
- **Web**
  - http://statmath.wu.ac.at/courses/data-analysis/itdtHTML/node55.html
  - https://en.wikipedia.org/wiki/Persistent_data_structure

32

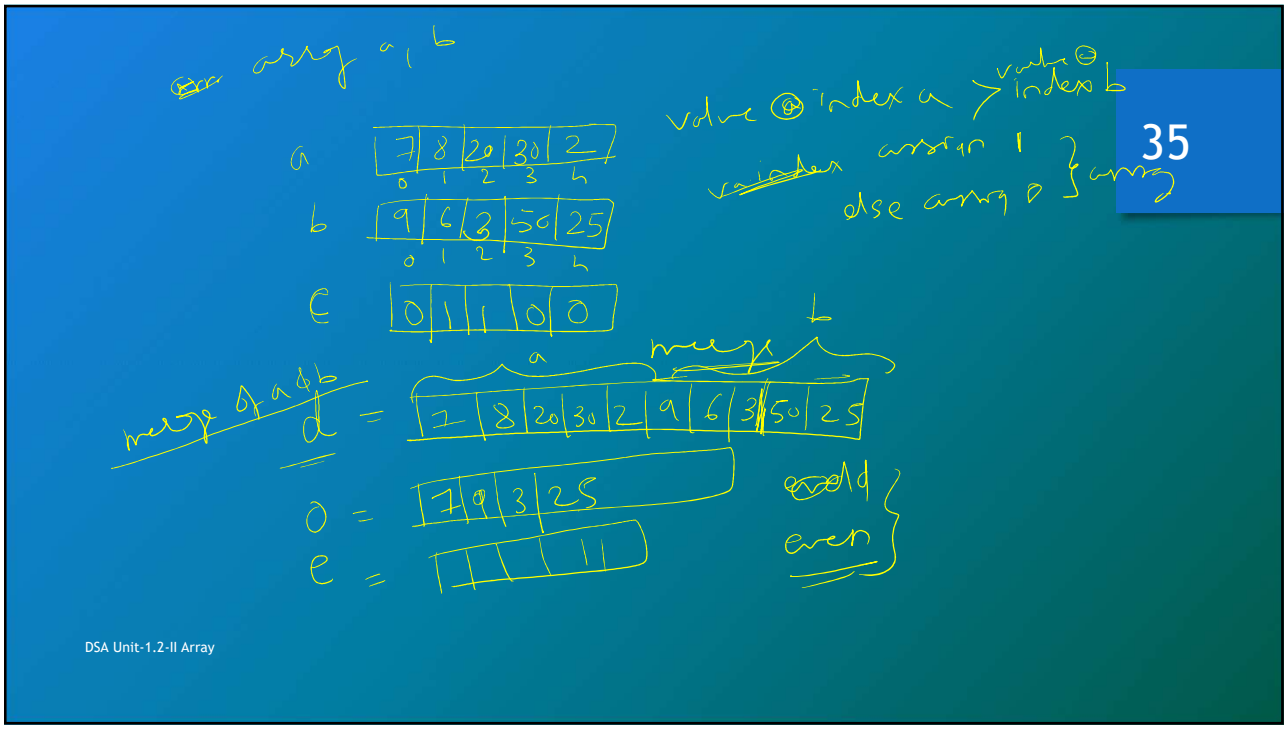
33

Thank You !!!

33



34



35