

CS 101: Computer Programming and Utilization

Jul-Nov 2016

Bernard Menezes
(cs101@cse.iitb.ac.in)

Lecture 10: Arrays

About These Slides

- Based on Chapter 14, 15 of the book *An Introduction to Programming Through C++* by Abhiram Ranade (Tata McGraw Hill, 2014)
- Original slides by Abhiram Ranade
 - First update by Varsha Apte
 - Second update by Uday Khedker

Computers Must Deal with Large Amounts of Data

Examples:

- Pressure measured at various points in an area
- Given altitudes of various points in a lake, find how much water is there given the water level.
- Account balance of thousands of bank customers
- Quiz 1 Marks of all CS 101 students

How to Handle Lot of Data?

- Fundamental problem: Writing out variable names to store each piece of data would be tiring
double pressure1, pressure2, ..., pressure1000;
- This is the problem solved using Arrays
- More elaborate, modern, and flexible solution involving **vector's** will be discussed later
- Arrays are simple to understand. Ideas useful in vectors too

Arrays

For storing a large amount of data of the same type

`double pressure[1000];`

- Essentially defines 1000 variables (**array elements**)
Variables are named `pressure[0]`, `pressure[1]`, `pressure[2]`,
..., `pressure[999]`
- The number inside [] is called **index**
- General form:

`data-type array-name[size];`

`array-name[i]` gives i^{th} variable (index is i here)

Necessary: $0 \leq i < \text{size}$. ($i \leq \text{size}-1$)

size also called **length**

Array Element Operations

- `double pressure[1000];`
- `cin >> pressure[0];`
- `for(int i=0; i<1000; i++)`
 `cin >> pressure[i];`
- `pressure[34] = (pressure[33]+pressure[35])/2;`
- `cout << pressure[439]*3.33 << endl;`

An array element is used in all the same ways as a **scalar** variable is used

Array index can be itself an expression which will be evaluated during execution and then the corresponding element will be used

Index Out of Range

```
double pressure[1000];  
pressure[1000] = 1.2;  
double d = pressure[-5];
```

In the assignments above, the array index is outside the allowed range: **0 through size-1**. In such cases the program may run and produce wrong results, may halt with a message. Nothing is guaranteed

The programmer must ensure index stays in range

Initialization While Declaring

```
int squares[4] = {0, 1, 4, 9};
```

```
int cubes[] = {0, 1, 8, 27, 64}; // size = 5 inferred.
```

```
int x, pqr[200], y[]={1,2,3,4,7};
```


Marks Display Problem

Read in marks of the 100 students in a class, given in roll number order, 1 to 100

After that, students may arrive in any order, and give their roll number. The program must respond by printing out their marks. If any illegal number is given as roll number, the program must terminate

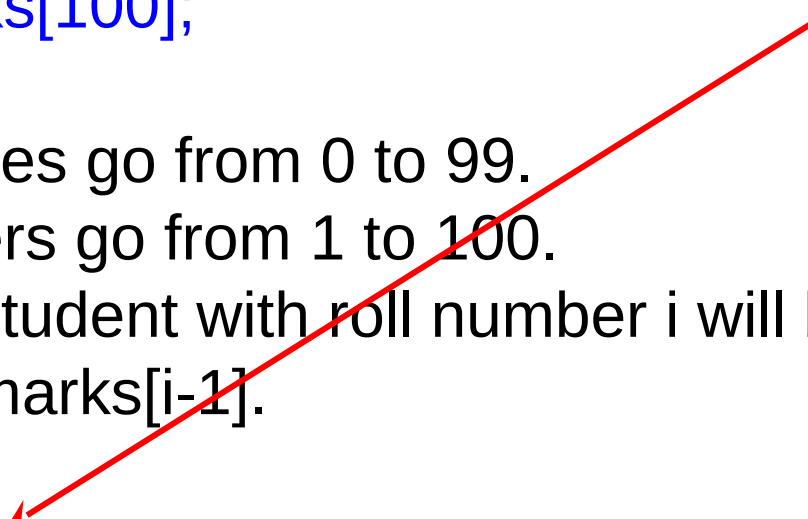
Program

```
double marks[100];

// array indices go from 0 to 99.
// roll numbers go from 1 to 100.
// marks of student with roll number i will be
// stored in marks[i-1].

for(int i=0; i<100; i++)
    cin >> marks[i];
while(true){
    int rollno;
    cin >> rollno;
    if(rollno < 1 || rollno > 100) break;
    cout << marks[rollno - 1] << endl;
}
```

Note the strictly
less than sign



Display Who Got Highest

Read marks as before. Display all roll numbers who got highest marks

```
// marks defined and read into as before.  
double maxsofar = marks[0];  
for(int i=1; i < 100; i++){  
    // Plan: in the ith iteration, maxsofar should  
    // hold the maximum of marks[0..i-1].  
    maxsofar = max(maxsofar, marks[i]);  
}
```

We can know the maximum marks only after seeing all the marks.

Hence identifying such students would need an additional iteration

Display Who Got Highest

```
// marks defined and read into as before.
```

```
double maxsofar = marks[0];
```

```
for(int i=1; i < 100; i++){
```

```
    // Plan: in the ith iteration, maxsofar should
```

```
    // hold the maximum of marks[0..i-1].
```

```
    maxsofar = max(maxsofar, marks[i]);
```

```
}
```

```
// maxsofar now holds max value in marks[0..99].
```

```
for(int i=0; i < 100; i++)
```

```
    if(marks[i] == maxsofar)
```

```
        cout << i+1 << endl; // Marks[i] holds marks of rollno i+1.
```

Accumulating the maximum into the variable **maxsofar**: Very standard idiom

Going over the array to filter elements that match a certain condition: also standard

Histogram

Read in marks as before, print how many scored between 1-10, 11-20, ..., 91-100

```
int hist[10];
```

```
// Plan: hist[i] will store number of students getting  
// marks between  $(10*i)+1$  and  $10*(i+1)$ 
```

On reading a certain mark v , add 1 to suitable element of `hist`

Which element? $(v-1)/10$, assuming v is integer, and truncation in division

Histogram

```
for(int i=0; i<10; i++) hist[i]=0;

for(int i=0; i<100; i++){

    double marks;

    cin >> marks;

    hist[ int(marks-1)/10 ]++;

    // int(..) converts to int.

}
```

Mark Display Variation

- Teacher enters 100 pairs of numbers: (rollno, marks),
- Roll numbers are not necessarily 1...100. **Can't become indices**
- Student types in roll number r. Program must print out marks if r is valid roll number
If r is -1, then stop
- Program idea: Store roll numbers into a **separate array**.
Examine each element of the array and see if it equals r. If so print corresponding marks from the **marks array**.

Linear Search of an Array

```
int rollno[100]; double marks[100];
for(int i=0; i<100; i++)
    cin << rollno[i] << marks[i];
while(true){
    int r; cin >> r;
    if(r == -1) break;
    for(int i=0; i<100; i++)
        if(rollno[i] == r){
            cout << marks[i] << endl;
            break;
        }
}
```


Polynomial Multiplication

- Given polynomials $A(x)$, $B(x)$
- $A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$
- $B(x) = b_0 + b_1x + b_2x^2 + \dots + b_mx^m$
- Need to find their product $C(x) = A(x) B(x)$
- $C(x) = c_0 + c_1x + c_2x^2 + \dots + c_{m+n}x^{m+n}$
- Given a_0, \dots, a_n and b_0, \dots, b_m find c_0, \dots, c_{m+n}
 - Natural to use an array of $n+1$ elements to store the coefficients of a degree n polynomial
- Algorithm idea:
 - Each term $a_i x^i$ in $A(x)$ will multiply each term $b_j x^j$ in $B(x)$ and the product $a_i b_j x^{i+j}$ will contribute to the term $c_{i+j} x^{i+j}$

Example of Degree 2 Polynomial

a: $2x^2 + x + 3$

Coefficients 2, 1, 3

b: $4x^2 + 5x + 6$

Coefficients 4, 5, 6

c: $8x^4 + 14x^3 + 29x^2 + 21x + 18$

Polynomial Multiplication

- Read the polynomials in two arrays a and b
(Read coefficient of degree i and store in ith index)
- Initialize all elements in array c to 0
- (Initially all coefficients in the result are 0)
- Implementing the Algorithm idea:
 - *Each term $a_i x^i$ in $A(x)$ will multiply each term $b_j x^j$ in $B(x)$ and the product $a_i b_j x^{i+j}$ will contribute to the term $c_{i+j} x^{i+j}$*
 - Multiply $a[i]$ with $b[j]$ and store in $c[i+j]$
 - Consider each i : $0 \leq i \leq \text{max_degree}$,
For each i consider each j : $0 \leq j \leq \text{max_degree}$

Program to Multiply Degree 10 Polynomials

```
double a[11], b[11], c[21];
// Polynomials A, B have degree 10, C has degree 20
for(int i=0; i<=10; i++)
    cin >> a[i]; // read in polynomial A
for(int j=0; j<=10; j++)
    cin >> b[j]; // read in polynomial B
for(int k=0; k<=20; k++)
    c[k] = 0;
for(int i=0; i<=10; i++) // Now multiply A and B
    for(int j=0; j<=10; j++)
        c[i+j] += a[i]*b[j]; // as discussed earlier.
for(int k=0; k<=20; k++)
    cout << c[k] << ' '; // output c, separated by spaces
cout << endl;
```

Dispatching Taxis

- Taxi drivers arrive: driverID put into “queue”. driverID : integer
- Customer arrives: If taxi is waiting, first driver in queue is assigned. If no taxi waiting, customer asked to call again later

Key Requirements

- Remember driverIDs of drivers who are waiting to pick up customers
- Remember the order of arrival
- When customer arrives: assign the earliest driver.
Remove driverID of assigned driver from memory
- When driver arrives: Add driver's driverID to memory

How to Remember DriverIDs

Use an array.

```
const int n=500;
```

```
int driverID[n];
```

n: maximum number of drivers that might have to wait simultaneously.

In what **order** to store the ids in the array?

What **other information** do we need to remember?

What do we do when customer arrives?

What do we do when driver arrives?

Idea 1

Store earliest driver in driverID[0]. Next earliest in driverID[1]. ...

Remember number of drivers waiting

```
int nWaiting;
```


Visualizing the Problem

driverID[]

Time	Driver Arrival	Customer Arrival
1	20	
2	14	
3	32	
4	3	A
5	5	
6	8	
7	22	B
8	21	
9	10	C

Program Outline

```
const int n=500; int driverID[n], nWaiting =
0;
while(true){
    char command; cin >> command;
    if(command == 'd'){
        // process driver arrival.
    }
    else if(command == 'c'){
        // process customer.
    }
    else if(command == 'x') break;
    else cout << "Illegal command.\n";
}
```

Invariants

- $n\text{Waiting}$ = number of waiting drivers.

Number of waiting drivers can be at most the array length

$$0 \leq n\text{Waiting} \leq n$$

- id of earliest waiting driver is in $\text{driverID}[0]$

Next in $\text{driverID}[1]$

...

Last in $\text{driverID}[n\text{Waiting}-1]$

Driver Arrival

```
if(nWaiting == n)
    cout << "Queue full.\n";
else{
    int d; cin >> d;
    driverID[nWaiting] = d;
    nWaiting ++;
}
```

When Customer Arrives:

Provided $n\text{Waiting} > 0$:

Assign the earliest unassigned driver to customer.

Earliest unassigned: stored in $\text{driverID}[0]$.

Second earliest should become new earliest...

Third earliest should become ...

$n\text{Waiting}$ should decrease.

Customer Arrival

```
if(nWaiting == 0)
    cout << "Try again later.\n";
else{
    cout << "Assigning " << driverID[0]
        << endl;
    for(int i=1; i <= nWaiting - 1; i++)
        driverID[i-1] = driverID[i];
    // Queue shifts up
    nWaiting-- ;
}
```

Idea 2

- Our program can be made more efficient.
- Emulate what might happen without computers.
- Names written on blackboard. Arriving driver IDs written top to bottom. When board bottom reached, begin from top if drivers have left.

Blackboard for Driver Dispatch

	DRIVER QUEUE	DRIVER QUEUE	DRIVER QUEUE	DRIVER QUEUE
0	657 ← First	657	546 ← Last	546
1	982	982		630
2	095	095		341 ← Last
3	457	457		
4	103 ← First	103 ← First	103 ← First	103 ← First
5	889	889	889	889
6	333	333	333	333
7	425	425	425	425
8	489	489	489	489
9	723	723	723	723
10	613	613	613	613
11	063	063	063	063
12	205 ← Last	205 ← Last	205	205

Assigned

Driver #546 arrives-Wrap around and start entering at the top

More Efficient Implementation

- Think of `driverID` as a circular array
- The next position after `driverID[n-1]` (bottom of board) is `driverID[0]` (top of board)

Invariants

- $n\text{Waiting}$ = number of waiting drivers

$$0 \leq n\text{Waiting} \leq n$$

- New variable front = position of earliest arriving driver who has not yet been assigned. front initialized to 0

$$0 \leq \text{front} < n$$

- Valid driver IDs are at

$$\text{driverID}[\text{front}] \dots \text{driverID}[(\text{front} + n\text{Waiting} - 1) \% n]$$

Note that $\%$ provides the effect of **wrapping around**

In the example (last table):

$$\begin{aligned} &\text{driverID}[4] \text{ to } \text{driverID}[(4+12-1)\%13] \\ &= \text{driverID}[4] \text{ to } \text{driverID}[2] \end{aligned}$$

Processing Driver Arrival

```
if(nWaiting == n)
    cout << "Queue full.\n";
else{
    int d; cin >> d;
    driverID[(front+nWaiting) % n] = d;
    nWaiting ++;
}
```

```
// front + nWaiting % L : index of
// empty position after end of queue.
```

Processing Customer Arrival

```
if(nWaiting == 0)
    cout << "Try later.\n";
else{
    cout << "Assigning " <<
        driverID[front] << endl;
    front = (front + 1) % n;
    nWaiting --;
}
```

Remarks

New idea is better, copying of elements of driverID is avoided.

Efficiency gain: Fixed number of operations

Exercise: make sure that the invariants indeed remain true after each customer or driver arrival.

Character Arrays

```
char name[10], address[50];
```

Can be used to store character sequences, or **character strings** (no longer than 9 or 49 resp.)

Typically we will not know the exact length of the name or the address

Standard protocol inherited from the C language:

- Store characters in the string in the array starting from element 0
- After all the characters are stored, store the character `'\0'`, sometimes called null (Character whose ASCII value is 0)
- Key idea: **everything until `'\0'` is a part of the string, not what comes later**
- No need to explicitly store the length of the string

Character Arrays With Initialization

```
char n1[20]="Ajanta", n2[]="Ellora";
```

This will create the arrays n1 and n2

n1 will be initialized as follows

- The first character string has length 6. So 'A', 'j', ... 'a' will get stored in n1[0] through n1[5]
- Finally '\0' will get stored in n1[6]
- The elements n1[7] through n1[19] will not be initialized

No length has been specified for n2. C++ will give it length 1 more than the length of the string "Ellora"

- It will also be initialized to the string "Ellora" followed by '\0'

Character Array Processing

Standard idiom: process the array from the beginning until '\0' is found.

```
bool findchar(const char *cstr, char x){  
    // determines if x occurs inside cstr  
    for(int i=0; cstr[i] != '\0'; i++){  
        if(cstr[i] == x) return true;  
    }  
    return false;  
}  
int main(){  
    char name[] = "India";  
    cout << findchar(name, 'd') << endl;  
    // should print 1;  
}
```

Key point to note: Array length is not passed to function, because it is not needed.

Reading and Printing char Arrays

```
char buffer[80];
```

```
cin >> buffer;
```

```
// Reads one word (space or newline terminated)
```

```
// and stores it into buffer, terminated by '\0'.
```

```
// If user types more than 80 characters, they will
```

```
// overflow buffer.
```

```
// unsafe.
```

```
cin.getline(buffer,80);
```

```
// Reads a line (terminated by newline) or at most 79
```

```
// characters. Stores them in buffer, terminated by '\0'.
```

```
// Safe.
```

```
cout << buffer;
```

```
// Prints the content of buffer till '\0'.
```

Concluding Remarks

- Later we will study the string class which is a much nicer and safer way of dealing with textual data
- When writing new code, use string, and not char arrays
- However you must know char arrays because basic ideas in string are similar. Also because char arrays are used a lot in C language code, which you may encounter

Two Dimensional Arrays

- Useful for storing matrices, or tables

```
double xyz[m][n];
```

- Creates $m \times n$ variables. The variables can be accessed by writing `xyz[i][j]`, where $0 \leq i < m$, and $0 \leq j < n$
- `xyz[i][0], xyz[i][1], ... xyz[i][n-1]` constitute i th **row** of `xyz`
- `xyz[0][j], xyz[1][j], ... xyz[m-1][j]` constitute j th **column** of `xyz`
- m, n are first and second **dimensions** of `xyz`
- Variables stored in memory in **row major** order, i.e. row 0, followed by row 1, ..., row $m-1$

Visualizing Two Dimensional Arrays

```
double xyz[m][n];
```

Two Dimensional Arrays

- Initialization possible

```
int pqr[2][3] = {{1,5,7}, {13,6,2}};
```

- Values picked up from the initialization list in row major order
- Enhanced versions of two dimensional arrays will be discussed later

Example 1

Create a 10x10 matrix A and initialize it to identity, i.e. value 1 in $A[i][i]$ for all i and 0 elsewhere

```
double A[10][10];
for(int i=0; i<10; i++)
    for(int j=0; j<10; j++)
        if(i == j)
            A[i][j] = 1;
        else
            A[i][j] = 0;
```

Example 2

- Create an array M to store marks of 10 students in 5 tests. Read the marks and store them in M.

```
double M[10][5];
for(int i=0; i<10; i++){
    cout << "Give marks of student " << i << ": ";
    for(int j=0; j<5; j++)
        cin >> M[i][j];
}
```

Example 3

```
char countries[3][20] = {"India", "Nepal", "China"};
```

- Creates array `countries` with 3 rows each containing 20 characters
- The rows are initialized as shown
- As usual, each character string is stored with a `'\0'` following it
- Individual characters can be accessed in the usual manner, e.g. `countries[1][1]` will be `'e'`
- Individual country can be accessed by writing `countries[i]`. This simply represents the string stored in row `i` of `countries`. The following will print out "China":

```
cout << countries[2] << endl;
```


Visualizing Example 3

```
char countries[3][20] = {"India", "Nepal", "China"};
```

Passing 2 Dimensional Arrays to Functions

- In the called function, the second dimension must be a constant
- The function will only work for two dimensional arrays having arbitrary number of rows, but each row having the specified size. This is a limitation

Passing 2 Dimensional Arrays to Functions

```
void printCountries(char c[][20], int n){
    for(int i=0; i<n; i++)
        cout << c[i] << endl;
}
int main(){
    char countries[3][20]= ... // as before
    printCountries(countries, 2);
    // will print out only first two countries
}
```