Arrays

Abhiram Ranade

Computers must deal with large amounts of data

- Simulate what happens when many balls are moving in a box. (Gas molecules?)
- Given altitudes of various points in a lake, find how much water is there given water level.
- Given the road map of India, find the shortest route from Kirloskarwadi to Tatanagar.

How to handle lot of data?

- Fundamental problem: Writing out variable names to store information would be tiring

```
double pressure1, pressure2, …, pressure1000;
```

- This is the problem solved using Arrays.

Arrays

```
double pressure[1000];
```

- Essentially defines 1000 variables ("array elements"). Variables are named pressure[0], pressure[1], pressure[2], …, pressure[999]
- General form:

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

array-name[index] gives indexth variable.

0 <= index < size.

Array element operations

```
double pressure[1000];
cin >> pressure[0];
for(int i=0; i<1000; i++) cin >> pressure[i];
// index can be an expression, which will be
// evaluated during execution.
pressure[34] = (pressure[33]+pressure[35])/2;
cout << pressure[439]*3.33 << endl;
```

Index out of range

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

In all 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 defining

```
int squares[5] = {0, 1, 4, 9, 16};
int cubes[] = {0, 1, 8, 27}; // size = 4 inferred.

int x, pqr[200], y[]={1,2,3,4,7,8,9};
```

size also called length.

Marks display problem

First 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]
for(int i=0; i<100; i++) cin >> marks[i];

while(true){
  int rollno; cout << “Roll no:”; cin >> rollno;
  if(rollno < 1 || rollno > 100) break;
  cout << marks[rollno – 1];  // why -1?
}
```

Display who got highest

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

```
// array marks defined and read in as before.
double maxsofar = marks[0];
for(int i=1; i < 100; i++)
    maxsofar = max(maxsofar, marks[i]);
for(int i=0; i < 100; i++)
   if(marks[i] == maxsofar) cout << i+1 << endl;
```

Histogram

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

```
int hist[10]; //hist[i] stores number of students
        // getting marks between 10*i+1 to 10*(i+1)
```

On reading a value v, add 1 to suitable element of hist.

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

Histogram

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

```
int hist[10]; //hist[i] stores number of students
              // getting marks between 10*i+1 to 10*(i+1)
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: rollno, marks.

```
int rollno[100];  // assume rollno is 8 digit int.
int marks[100];
```

- Student types in roll number r. Program must print out marks if r is valid roll number.
- Program idea: search rollno array to see if r is present. If so print corresponding marks.

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++) // look at each element
      if(rollno[i] == r) cout << marks[i] << endl;
} // Can you make a small change to improve?
```

Polynomial Multiplication

$A(x) = a_0 + a_1x + a_2x^2 + \ldots.+ a_nx^n$

$B(x) = b_0 + b_1x + b_2x^2 + \ldots.+ b_mx^m$

$C(x) = A(x)\,B(x) = c_0 + c_1x + c_2x^2 + \ldots.+ c_{m+n}x^{m+n}$

Given $a_0, \ldots, a_n$ and $b_0, \ldots, b_m$ find $c_0, \ldots, c_{m+n}$.

- Strategy 1: Multiply A(x) by b_jx^j for all j. Accumulate products into result.
- Strategy 2: Find formula for c_k.

Strategy 2: Multiply A(x) by $b_j x^j$

$A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$

$B(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_m x^m$

$C(x) = A(x)\,B(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{m+n} x^{m+n}$

Given $a_0, \ldots, a_n$ and $b_0, \ldots, b_m$ find $c_0, \ldots, c_{m+n}$.

$A(x) * b_j x^j = a_0 b_j x^j + a_1 b_j x^{j+1} + \ldots + a_n b_j x^{n+j}$

Add $a_i b_j$ to c_{i+j}.

Program to multiply degree 10 polynomials

```
double a[11], b[11], c[21];  // c has degree 20.
for(int i=0; i<=10; i++) cin >> a[i];
for(int j=0; j<=10; j++) cin >> b[j];
for(int k=0; k<=20; k++) c[k] = 0;
for(int j=0; j<=10; j++)     // for each b_ j x^j
  for(int i=0; i<=10; i++) // multiply A(x)
    c[i+j] += a[i]*b[j];    // calculate product term
                            // and add immediately
```

Strategy 2: Formula for c_k

$A(x) = a_0 + a_1x + a_2x^2 + \ldots.+ a_nx^n$

$B(x) = b_0 + b_1x + b_2x^2 + \ldots.+ b_mx^m$

$C(x) = A(x)\,B(x) = c_0 + c_1x + c_2x^2 + \ldots.+ c_{m+n}x^{m+n}$

Given $a_0, \ldots, a_n$ and $b_0, \ldots, b_m$ find $c_0, \ldots, c_{m+n}$.

c_k = coefficient of x^k, contributions from products a_ib_j where $i+j = k$.

$c_k = a_0b_k + a_1b_{k-1} + \ldots + a_kb_0$, provided $k <= m, n$.

for larger k: sum only over valid coefficients.

Program to multiply degree 10 polynomials

```
double a[11], b[11], c[21];  // c has degree 20.
for(int i=0; i<=10; i++) cin >> a[i];
for(int j=0; j<=10; j++) cin >> b[j];
for(int k=0; k<=20; k++){
   c[k] = 0;
   for(int i = 0; i<= k; i++)
      if(i < 11 && k-i <11) c[k] += a[i] * b[k – i];
} // compute term only if coefficients are valid
```

Dispatching Taxis

- Taxi drivers report: driverID put into "Queue".
- driverID : integer

- Customer arrives. If taxi is waiting, first 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.

long long int driverID[500];

- Length: largest number of drivers you expect will be waiting.
- 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;
```

Outline

```
long long int driverID[500];    int 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

- nWaiting = number of waiting drivers.
- 0 <= nWaiting <= 500
- Earliest waiting driver is at driverID[0]. Next at driverID[1], …

Driver arrival

```
if(nWaiting == 500) cout << “Queue full.\n”;
else{
  long long d; cin >> d;
  driverID[nWaiting] = d;
  nWaiting ++;
}
```

When customer arrives:

Provide nWaiting > 0:

- Assign the earliest unassigned driver to customer. Earliest unassigned: stored in driverID[0].
- Second earliest should become new earliest…
- Third earliest should become …
- nWaiting 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

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.
- Think of driverID as a circular array. "Next" position after driverID[499] (bottom of board) is driverID[0] (top of board).

Invariants

- nWaiting = number of waiting drivers.
- 0 <= nWaiting <= 500
- New variable front = position of earliest arriving driver who has not yet been assigned. front initialized to 0.
  - 0 <= front < 500
- Valid driver IDs are at driverID[front] … driverID[(front + nWaiting – 1) % 500]

Processing driver arrival

```
if(nWaiting == 500) cout << “Queue full.\n”;
else{
  long long d; cin >> d;
  driverID[(front + nWaiting) % 500] = d;
  nWaiting ++;
}
```

Processing Customer Arrival

```
if(nWaiting == 0) cout << “Try later.\n”;
else{
   cout << “Assigning “ << driverID[front] << endl;
   front = (front + 1) % 500;
   nWaiting --;
}
```

A geometric Problem

Given centers and radii of n circles, determine if any circles intersect.

- Read in center coordinates and radius of each.
- Consider each circle and check if it intersects with others.
- C1, C2 intersect if distance between centers <= sum of radii.

Program

```
double x[10], y[10], r[10];  bool intersect = false;
for(int i=0; i<10; i++) cin >> x[i] >> y[i] >> r[i];
for(int i=0; i<10; i++)
  for(int j=0; j<10; j++)
    if(pow(x[i]-x[j],2)+pow(y[i]-y[j],2)
       <= pow(r[i]+r[j], 2)) intersect = true;
```