An Introduction to Programming though C++

Abhiram G. Ranade

Lecture 7.1

Ch. 14: Arrays

Mark display variation

- Roll numbers are not in range 1 .. 100, but a larger range, e.g. 170010022.
- Marklist = 100 pairs of numbers: (rollno, marks), … .
- Teacher must enter roll number, marks into the computer.
- Later:
  - Students arrive and each 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:

- Use an additional array for storing roll numbers.
- Store ith roll number in rollno[i] and ith mark into marks[i].
- When students arrive: Examine each element of the roll number array and see if it equals ***r***. If so print corresponding marks.

The program

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

while(true){
  int r; cin >> r;      // read in query roll number
  if(r == -1) break;
  bool found = false;
  for(int i=0; i<100; i++){
    if(rollno[i] == r){
      cout << marks[i] << endl;
      found = true;
      break;
    }
  }
  if(!found) cout << "Roll number not found.\n";
}
```

Demo

- generalRollNos.cpp

Exercise

- Modify the program so that there are marks for two subjects.

What we discussed

- If the roll numbers are not in the range 0..N-1 where N is the number of students, then we need to store the roll numbers also.
- Searching through the array to find data corresponding to a “key” is a common idiom.
  - Use of a “found” variable is a natural strategy

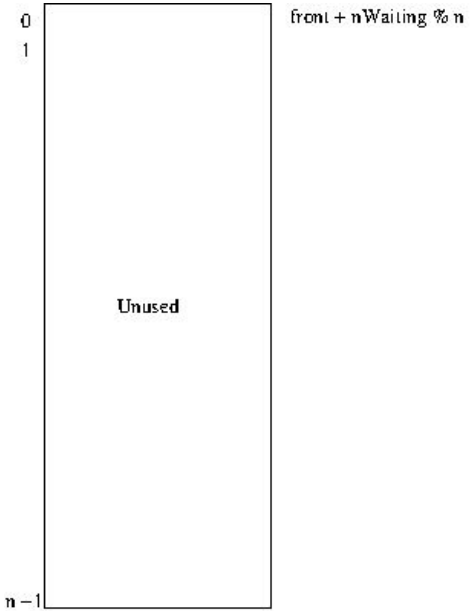
Next: Polynomial multiplication

Taxi Dispatch: Idea 2

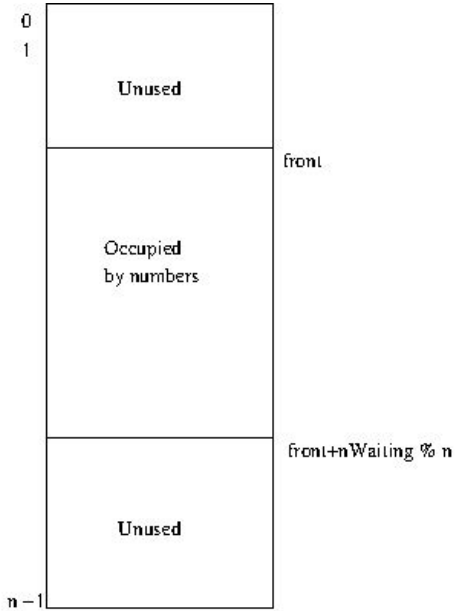
- Previous program seems to be copying data a lot...
  - Can we avoid the copying?
- Emulate what might happen without computers.
  - Names written on blackboard.
  - Arriving driver IDs written top to bottom.
  - When board bottom reached, begin from top if drivers have left.
  - Think of ***driverID*** as a circular array.
  - "Next" position after ***driverID[n-1]*** (bottom of board) is ***driverID[0]*** (top of board).

Invariants

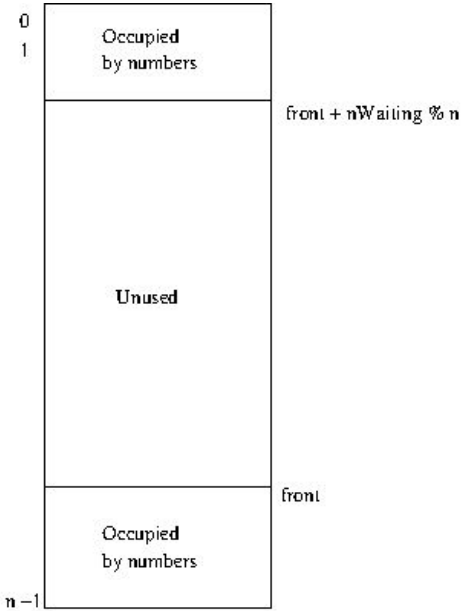
- ***nWaiting*** = number of waiting drivers. ***0 <= nWaiting <= n***
- New variable ***front*** : position of earliest unassigned driver ***= 0*** initially.
  0 <= front < n
- First valid driver ID is at ***driverID[front]***
- Where is the next? ***driverID[front+1]?***
  - If ***front = n-1,*** then ***front+1 = n,*** which is not a valid index.
  - Next waiting driver is ***driverID[(front+1) % n]***
  - Last waiting driver id is: ***driverID[(front+nWaiting—1) % n]***
- Note that **%** provides the effect of “wrapping around”.



(a) At the beginning



(b) After some time



(a) After more time

Processing driver arrival

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

// front + nWaiting % n : 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 --;
}
```

What we have discussed

- Different idea for solving taxi dispatch problem.
- New idea is better, copying of elements of ***driverID*** is avoided.
- Exercise: verify that the invariants indeed remain true after each customer or driver arrival.

Next: Disk intersection problem

Disk intersection problem

- Given a collection of n disks in the plane, decide if any two intersect.

Algorithm: check all possible pairs.

- Let i = index of smaller numbered disk.
- So i must range from 0 to n-2
- For each i, the other disk must range from i+1 to n-1.

Intersection check:

- Distance between centers must be less than sum of the radii.

Program

```
const int n=5;
double centerx[n], centery[n], radius[n];
for(int i=0; i<n; i++)  cin >> centerx[i] >> centery[i] >> radius[i];

bool intersect = false;
for(int i=0; i<n-1; i++){
   for(int j=i+1; j<n; j++)
      if(pow(centerx[i]-centerx[j],2)+pow(centery[i]-centery[j],2) < pow(radius[i]+radius[j], 2)){
         intersect = true;
         break;
      }
 }
 cout << intersect << endl;
```

What we have discussed

- Given a set of objects, how do you go over all pairs.
- Exercise: Can you generalize the idea?
  - Write a program to check if any 3 of a given set of n points in the plane are collinear.

Next: arrays of graphics objects, conclusion of lecture sequence

Arrays of graphics objects

- We can create multiple graphics objects by using arrays.

```
Turtle t[3];
T[1].right(120);
```

A program

```
int main(){
  initCanvas();
  Turtle t[3];

  t[1].left(120);
  t[2].left(240);

  repeat(8){
    for(int i=0; i<3; i++) t[i].forward(100);
    for(int i=0; i<3; i++) t[i].left(360.0/8);
  }
  wait(5);
}
```

Demo

- 3poly.cpp

Arrays: concluding remarks

- Way to store many objects of the same type in memory, without having to separately define a variable for each.
- Index: used to choose an element.
  - Must be at least zero and at most array length – 1.
  - Can be an expression
- Index may sometimes play an active role:
  - When roll numbers are consecutive from 0 to N-1
  - Polynomial representation: ith coefficient stored in ith element
  - Taxi dispatch: indicates arrival order
- Index may not have significance:
  - When roll numbers took on arbitrary values
- Indexing into an array happens very fast, independent of how many elements are present in the array.

More remarks

Several idioms used with arrays:

- Scan through all elements to find a matching element (“linear search”)
- Do some calculation which determines the index where something is to be done (“histogram”)
- Use the array like a queue of elements which wait for something to happen, after which they leave the queue (“taxi dispatch”)