CS 101:
Computer Programming and Utilization
Lecture 8: Arrays

Jul-Nov 2017

Umesh Bellur
(cs101@cse.iitb.ac.in)

Computers Must Deal with Large Amounts of (related) Data

Examples:

- Pressure measured at various points in an area

- Given depth 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

- A string of characters – names, addresses, sentences

How to treat this collection of similar 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

Arrays

For storing a large amount of data of the same type

`double pressure[1000];`

• **Defines** `1000 variables (``array elements``) Variables are named pressure[0], pressure[1], pressure[2], ···, pressure[999]`

• **The number inside [ ] is called the array** index

• `General form:`

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

    `array-name[i] gives i`[th] `variable` **(index is i here)**

    `Necessary: 0 <= i < size.` **(i <= 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 **itself be** 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
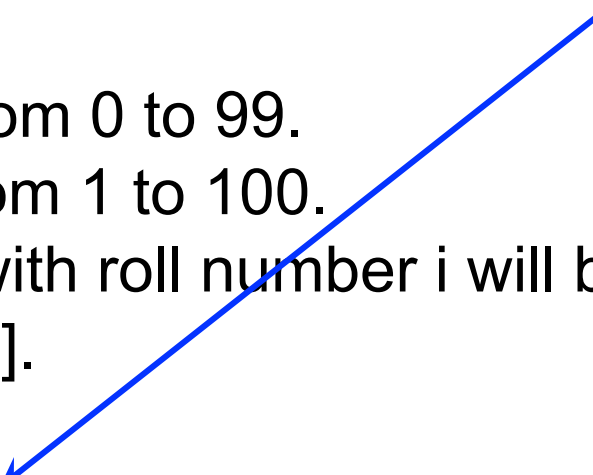
Program

```cpp
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++){
   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.
```

Frequency Distribution Histogram

Read in marks as before, print how many scored between 1-10, 11-20, ..., 91-100
What kind of array do we need?

```
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

```cpp
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 $A(x)$, $B(x)$
- $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$
- $B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_mx^m$
- Need to find their product $C(x) = A(x)\,B(x)$
- $C(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{m+n}x^{m+n}$
- Given $a_0, \ldots, a_n$ and $b_0, \ldots, b_m$ find $c_0, \ldots, c_{m+n}$
  - Natural to use an array of n+1 elements to store the coefficients of a degree n polynomial
- Algorithm idea:
  - Each term a_ix^i in $A(x)$ will multiply each term b_jx^j in $B(x)$ and the product $a_ib_jx^{i+j}$ 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: $8x^4 + 14x^3 + 29x^2 + 21x + 18$

Program to Multiply Degree 10 Polynomials

```cpp
// Polynomials A, B have degree 10, C has degree 20
double a[11], b[11], c[21];
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] << ' ';
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";
}
```

Driver Arrival

```cpp
if(nWaiting == n)

   cout << "Queue full.\n" ;

else {

   int d; cin >> d;

   driverID[nWaiting] = d;

   nWaiting ++;

}
```

When Customer Arrives:

Provided 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 ...
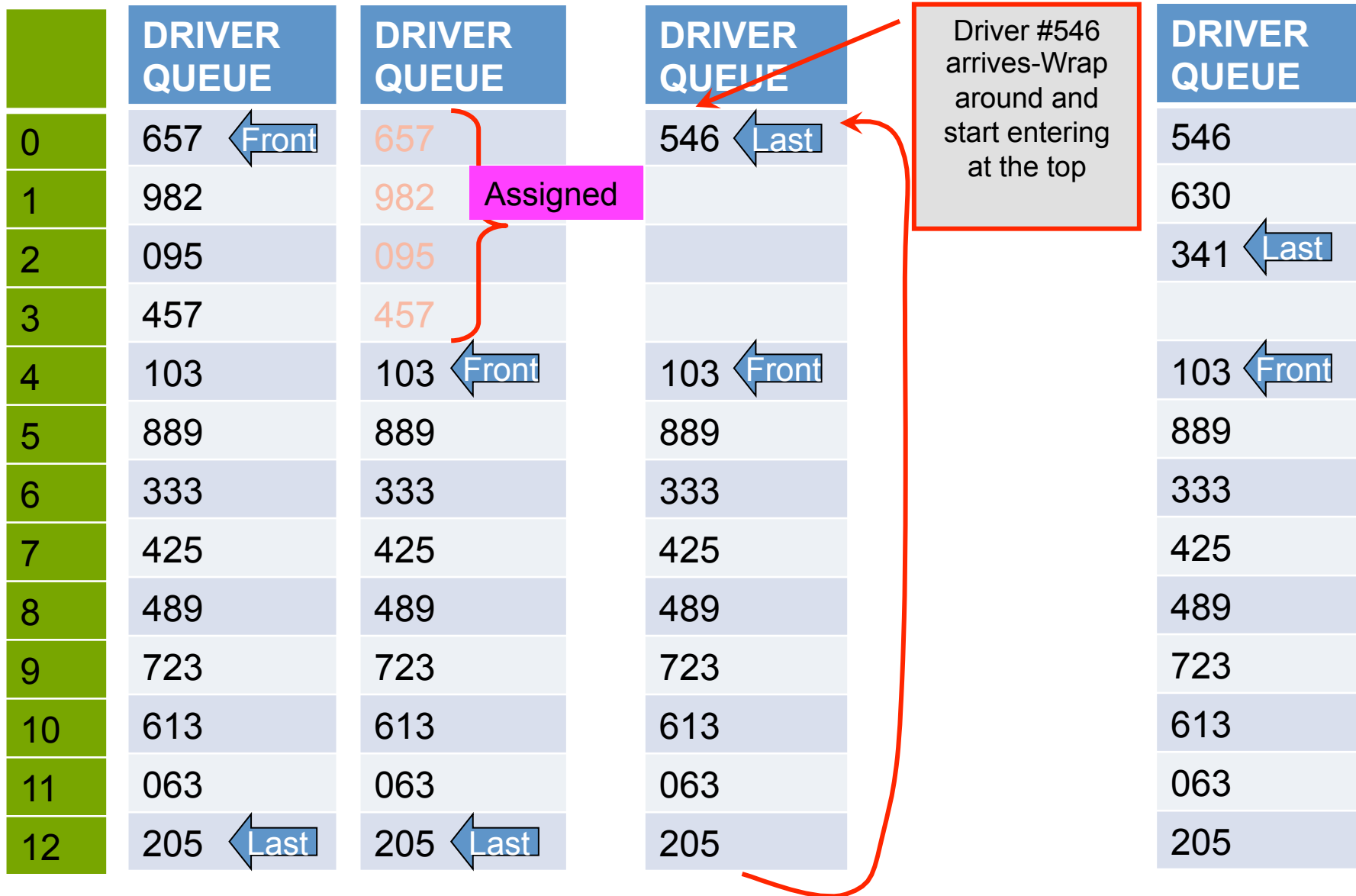
nWaiting should decrease.

Customer Arrival

```cpp
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 p`rogram 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 ◁ Front	657	546 ◁ Last		546
1	982	982			630
2	095	095			341 ◁ Last
3	457	457			
4	103	103 ◁ Front	103 ◁ Front		103 ◁ Front
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 **queue**

- **The n**ext position after driverID[n-1] (bottom of board) is driverID[0] (top of board)

Processing Driver Arrival

```cpp
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 --;

}
```

Textual data and arrays

- `char` type meant to store single letter.

- Array of `char` can be used to store sequences or letters, e.g. words, sentences, paragraphs.

Some operations with Character Arrays

```cpp
char* pA="abc";
char A[3]={'a','b','c'};

cout << "pA is " << pA << endl; // It displays "abc"

// The next may display wrongly because it does not know what is the
// last element
cout << "A is " << A << endl;

char B[]={'a','b','c','\0'}; // added a null character at the end
                            // contents of B is now similar to pA
cout << "B is " << B << endl; // It displays "abc"

int lengthpA = strlen(pA); // strlen is a function to know the lenght
cout << "length pA is " << lengthpA << endl; // displays 3
int lengthA = strlen(A);
cout << "length A is " << lengthA << endl;
// may display the size wrongly though we have set size to 3
// most of the size calculations needs a null character to indicate
// the end of the array
// null character is not counted in the size
```

Constness & character arrays

```
char y[] = "this is cs101"; // y is now const.
y = "this is so wrong"; // compiler error.

char x[5];
x = "abc"; // compiler error
x[3] = 'd'; // will compile.
```

Two Dimensional Arrays

- Useful for storing matrices, or tables

  `double xyz[m][n];`

- Creates m*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 ith row of xyz
- `xyz[0][j], xyz[1][j], … xyz[m-1][j]` constitute jth column

- 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*

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

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.

```cpp
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];
}
```

Outline

- The computer's view of arrays. This will help us better understand:
  - Where are the elements stored in memory
  - What happens when an index out of range is used
  - Function calls using arrays

- A function for sorting an array.
  - Sort: rearrange elements so that they are in non-decreasing or non-increasing order

Computer's view of array definition

```
int q[4] = {11,12,13,14};
```

- Assuming a single `int` uses one four byte word, 4 consecutive words of memory are allocated to q.

- Allocated memory used to store the variables `q[0]`, `q[1]`, `q[2]`, `q[3]` in order.

- Initial values stored.

Possible outcome

Address	Used for	Content
1004	q[0]	11
1008	q[1]	12
1012	q[2]	13
1016	q[3]	14

"Address": address of first byte.

Address 1004: bytes 1004, 1005, 1006, 1007

Computer's interpretation of array name

- Array name = address of allocated block = address of 0th array element.

- For `int q[4];` defined earlier:

- `cout << q ;` will print address 1004
  - Addresses are printed in hexadecimal. "0x12…"

- Type of q : int *

- Array name is a pointer, but its value cannot be changed. "q = 1008" is illegal.

In general

`elemtype aname[alength];`

- Block of memory of length `S*alength` is allocated, where S = size in bytes of a single `elemtype` variable.

- `aname` = starting address of zeroth element = address of allocated block. Const.

- Type of `aname`: `elemtype *`

- Type of `aname[i]` : `elemtype`

How does the computer interpret aname[index]

- [] is a binary operator - aname, index are the operands.

- aname[index] means
  - The variable stored at aname + S * index, where S = size of a single element of the type aname points to.
  - Example: for aname = q as before: S = 4
  - Yes, the computer does a multiplication and addition to find the position of the element in memory.
  - Note that only a single multiplication and addition is done, however large the array is.

Example

Our old array q

```
int q[4];
```

Address   Used for

1004-7   q[0]

1008-11   q[1]

1012-15   q[2]

1016-19   q[3]

q = 1004

type of q = int*

Computer's view of q[3]

- `q[3]` : variable of the type that q points to, stored at address q + `S*3` where `S` is size of a single variable of the type that q points to.

- variable of type `int`, stored at 1004 + 3*4 = 1016.

- Same as what we call `q[3]`

Index out of range

Our old array q

`int q[4];`

Address	Used for
1004-7	`q[0]`
1008-11	`q[1]`
1012-15	`q[2]`
1016-19	`q[3]`

q = 1004

type of `q = int*`

Suppose we execute:

`q[10] = 34;`

- `q[10]` : Mechanical interpretation as per our rule: variable of the type that q points to, stored at address `q + 10*S` where `S` is size of a single variable of the type that `q` points to.

- variable of type `int`, stored at 1004 + 10*4 = 1044.

- 34 will get stored in address 1044 which is not part of `q`!

- Possibly some other variable will be written into!

(contd.)

- `x = q[10]` : `x` will get some strange value.
- The computer may forbid accessing some addresses, if 1044 is such an address, computer may halt with an error message!
- It's the programmer's responsibility to ensure that index is in correct range!

Index checking

- Some programming languages prevent index out of range by explicitly checking.

  - First the value of the index is checked to see if it lies in the range 0..size-1. If it does then the address is calculated; if not an error message is printed and the program stops.

- Index checking is not done because it takes extra work, and because C++ designers believe that it is the programmer's job to ensure that the index is in range.
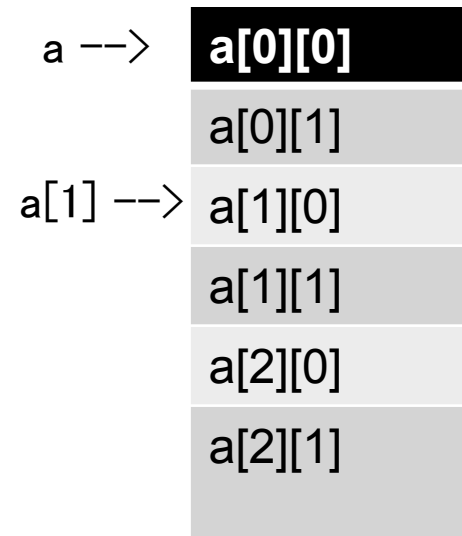
Exercise

- What does the following code do?

```
int q[4]={0,0,0,0};
int *r;
r = q;
r[3] = 5;
cout << q[3] << endl;
```

Two-dimensional array in memory

- Stored in row-major format.
- a[3][2]

a -->	**a[0][0]**
	a[0][1]
a[1] -->	a[1][0]
	a[1][1]
	a[2][0]
	a[2][1]

Function calls & array parameters

We might like to write functions to:

- find the largest value in the array

- find whether a given value is present in the array.

- find the average of the elements in the array.

- ...

Standard protocol of function calls

- Non array arguments are copied from activation frame of caller to the activation frame of called function.

Should complete array be copied?

- Arrays might be large, so might take very long. Seems like a waste of time.

- C++ does not support this.

How arrays are passed to functions

- Convention (You must do this):
  - Do not copy array elements.
  - Pass two arguments (a) starting address A (b) number of elements, n.
- Can elements be accessed in called function?
  - The expression A[i] can be used in the called function to access the ith element because of how [] works.

A program to find the average of elements in array

```
double avg(double M[], int n)
{
  double sum = 0;
  for(int i=0; i<n; i++)
    sum +=M[i];;
  return sum/n;
}
int main(){
  double q[]={11,12,13,14};

  cout << avg(q, 4) << endl;
}
```
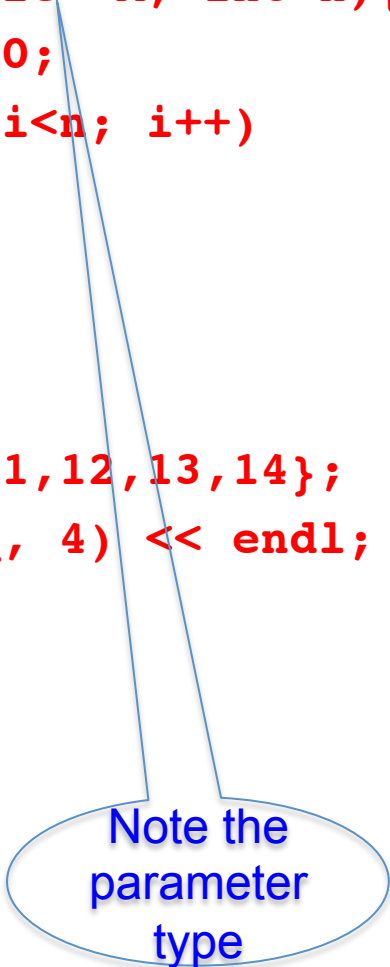
Note the parameter type

- Let us first check if this is a syntactically valid program, never mind what it does.
- The types of the arguments to a call must match the types of the parameters.
- The first parameter of avg has type double[], the first argument in the call is q, whose type is double[], because it points to the first element of a double array.
- The second parameter is of type int, and 4 in the call is indeed an int.

Equivalent call...

```
double avg(double* M, int n){
    double sum = 0;
    for(int i=0; i<n; i++)
        sum +=M[i];
    return sum/n;
}
int main(){
    double q[]={11,12,13,14};
    cout << avg(q, 4) << endl;
}
```

Note the parameter type

- Let us first check if this is a syntactically valid program, never mind what it does.
- The types of the arguments to a call must match the types of the parameters.
- The first parameter of avg has type double*, the first argument in the call is q, whose type is double*, because it points to the first element of a double array.
- The second parameter is of type int, and 4 in the call is indeed an int.

A program to find the average of elements in array

```
double avg(double M[], int n)
{
  double sum = 0;
  for(int i=0; i<n; i++)
    sum +=M[i];
  return sum/n;
}
int main(){
  double q[]={11,12,13,14};
  cout << avg(q, 4) << endl;
}
```

- On execution of avg(q,4) in main
- Activation frame created for avg.
- Value of q (starting address of array) copied into parameter M.
- Thus M[i] in avg means q[i] of main.
- Thus average of the elements of q is calculated in avg.
- The average is returned, and printed.

Remarks

- The function call mechanism is just call by value; the value of the **array name** is copied over. Nothing special is needed.

- The interesting part is the [ ] operator: given an address of an array and an index it can get us to the corresponding element, even if the address belongs to a different activation frame.

- The second argument to avg is not "required" to be the array length. If it is smaller, then the function will return the average of just that part of the array.

Passing 2 Dimensional Arrays to Functions

```cpp
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
}
```

Passing part of a 2 Dimensional Arrays to Functions

```cpp
int sum(int c[], int n){
    int s = 0;
    for(int i=0; i<n; i++)
        s += c[i];
    return s;
}
int main(){
    int matrix[3][20]= … // as before
    cout << "Sum of second row "
        << sum(matrix[1],20);
}
```

Sorting an array

- Suppose we are given an array containing numbers.

- We want to rearrange the numbers so that they appear in non-decreasing order.

- Example:

- Array initially: 35, 12, 29, 70, 18, 29

- Desired order: 12, 18, 29, 29, 35, 70

Sorting an array (contd.)

- Sorting is an important operation. Chapter 16 gives a clue why.

- There are many algorithms for sorting. Chapter 16 will discuss a clever and fast algorithm.

- Here we discuss a slow, but easy to understand algorithm: Selection Sort.

Selection Sort Algorithm

- Find the largest number.

- Exchange it with the element in the last position.

- We have made progress, because the last position now contains the largest, as we would like it to.

- Now we can apply the same idea to the first N-1 elements of the array, where N = length of the array.

- Then to first N-2 elements, and so on.

Finding the index of the largest element

```
int argmax(float data[], int L){
  // Scan the array from index 0 to L-1.
  // At all instants keep the index of the largest
  // found so far in a variable maxIndex.
  // Invariant for iteration i: maxIndex will be
  // the index of the max in data[0..i-1].
  int i=1, maxIndex=0;   // invariant holds.
  for(i=1; i<L; i++)
    if(data[maxIndex] < data[i])
         maxIndex = i;
  return maxIndex;
}
```

The main function and main program

```
void selSort(float data[], int N){
  for(int i=N; i>1; i--){
    int maxIndex = argMax(data,i);
    // Returns index of the largest in data[0…i-1]
    float maxVal = data[maxIndex];
    data[maxIndex] = data[i-1];
    data[i-1] = maxVal;          // exchange done!
  }
}
int main(){
  float a[6] = {35, 12, 29, 70, 18, 29};
  selsort(a, 6);
}
```