An Introduction to Programming through C++

Abhiram Ranade

Lecture 7.2

Ch 14: Arrays, part 2.

Function calls on arrays

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.
- …

We see how to write functions involving arrays next.

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;
}
```

- 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***.
- In each iteration, ***M[i]*** is needed.
- ***M*** has value = starting address of ***q***, and type ***double****
- ***M[i]*** means "Variable of type ***double*** at address ***q + 4*i***"
- 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.

How arrays are passed to functions

Pass two arguments:

- Starting address of the array, corresponding parameter is A, suppose.
- Number of elements in the array, 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.
- Array elements are not copied. If the function modifies the elements, then the modification will be seen in the calling program.

Remarks

- When you pass the name of an array in a function call, its value, the starting address of the array, is copied to the corresponding parameter.
- But because we pass the starting address of the array, we are effectively enabling the function to pass the array.
- Can say
  - “Array name is passed by value”
  - “Array elements are passed by pointer”
- The interesting part is the ***[]*** operator: given the 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.

Remarks

- An alternate syntax is also allowed.

int avg(double M[], int n){...}

- In this ***double M[]*** is synonymous with ***double* M***, but slightly more indicative that we expect ***M*** to be an array name, and not just any old pointer.
- 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.

What we discussed

- No special mechanisms are needed to pass an array to a function.
- We pass a pointer to the zeroth element, i.e. we state where the array begins in memory.
- We pass the number of elements.
- The [] operator can be used to access the individual elements of the array.
- Next: Function to sort an array.

🚲

Sorting an array

- **Input**: an array containing numbers.
- **Goal**: 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

Basic idea:

- Find the largest number.
- Exchange it with the element in the last position.
- We have made progress:
    - 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 posOfMax(float A[], int L){
// Returns the index of the largest element in A.
// Invariant for iteration i:
//      maxIndex = the index of the max in A[0..i-1].
  int i=1, maxIndex=0;  // invariant holds.
  for(i=1; i<L; i++)
    if(A[maxIndex] < A[i]) maxIndex = i;
  return maxIndex;
}
//Number of comparisons: L-1
```

The main function and main program

```
void selSort(float A[], int N){
  for(int i=N; i>1; i--){
    int maxIndex = posOfMax(A,i);  // posOfMax thinks array only has i elements
                              // exchange A[i-1], A[maxIndex]
    float maxVal = A[maxIndex];
    A[maxIndex] = A[i-1];
    A[i-1] = maxVal;             // exchange done!
  }
}
int main(){
  float a[6] = {35, 12, 29, 70, 18, 29};
  selSort(a, 6);
}
```

Remarks

- ***posOfMax*** will perform ***L-1*** comparisons, where L = length of array passed to ***posOfMax***.
- ***selSor***t calls ***argMax*** for ***L = N, N-1, …, 2***
- So the number of comparisons =

$$N + N-1 + \ldots + 2 = (N+2)(N-1)/2 \approx N^2/2$$

- There exist better algorithms which perform far fewer comparisons and thus run faster.
- Demo selSort.cpp

Variation

Express selsort as a recursive program.

Recursive selsort

```
void RselSort(float A[], int N){
  if(N == 1) return; // base case
  int maxIndex = posOfMax(A,N);
  float maxVal = A[maxIndex];
  A[maxIndex] = A[N-1];
  A[N-1] = maxVal;   // exchange
  RselSort(A, N-1);
}
```

Demo

RselSort.cpp

Concluding remarks

- The name of an array can be used in a program without the index, its value is the address of the 0th element of the array.
- Indexing into an array is to be thought of as an operation [] between the array name and the index.
- We can “pass” an array to a function by passing the array name. This does not copy the values but merely copies the address of the 0th element. In addition the length of the array must also be passed.
- Writing functions on arrays is useful, and this is a skill you must master.
- Selection sort runs in time proportional to the square of the number of elements being sorted. Faster algorithms are discussed in Chapter 16.