An Introduction to Programming

through C++

with

Manoj Prabhakaran

Lecture 12

Arrays

Based on material developed by Prof. Abhiram G. Ranade

Example: Counting

- To read in a word (lower case) and count the occurrences of each vowel

```
#include <iostream>
using std::cout; using std::cin; using std::endl;
int main() {
  int Na=0, Ne=0, Ni=0, No=0, Nu=0; char c;
  for(cin >> c; c >= 'a' && c <= 'z'; cin >> c) {
    if (c == 'a') ++Na;
    else if (c == 'e') ++Ne;
    else if (c == 'i') ++Ni;
    else if (c == 'o') ++No;
    else if (c == 'u') ++Nu;
  }
  cout << "#a = " << Na << ", #e = " << Ne << ", #i = " << Ni
                      << ", #o = " << No << ", #u = " << Nu << endl;
}
```

But, what if you wanted to count the occurrences of each character in the alphabet?

26 counters!

Example: Palindrome

- Read in an 8-letter word and check if it is a palindrome

```
#include <iostream>
using std::cout; using std::cin; using std::endl;
int main() {
  bool palindrome = true;
  char c1, c2, c3, c4, next;
  cin >> c1 >> c2 >> c3 >> c4;
  if ( (cin >> next, next != c4)
          ||  (cin >> next, next != c3)
          ||  (cin >> next, next != c2)
          ||  (cin >> next, next != c1) )
      cout << "Not a palindrome!" << endl;
  else
      cout << "Palindrome!" << endl;
}
```

But, what if you wanted to check a 100 letter string?

50 variables!

Example: Sorting

- Read in 3 numbers and print them in order

```
#include <iostream>
using std::cout; using std::cin; using std::endl;
int main() {
  int a1, a2, a3;
  cin >> a1 >> a2 >> a3;
  if (a1 <= a2 && a2 <= a3)
      cout << a1 << ", " << a2 << ", " << a3 << endl;
  else if (a1 <= a3 && a3 <= a2)
      cout << a1 << ", " << a3 << ", " << a2 << endl;
  else if (a2 <= a3 && a3 <= a1)
      cout << a2 << ", " << a3 << ", " << a1 << endl;
  else if (a2 <= a1 && a1 <= a3)
      cout << a2 << ", " << a1 << ", " << a3 << endl;
  else if (a3 <= a1 && a1 <= a2)
      cout << a3 << ", " << a1 << ", " << a2 << endl;
  else if (a3 <= a2 && a2 <= a1)
      cout << a3 << ", " << a2 << ", " << a1 << endl;
}
```

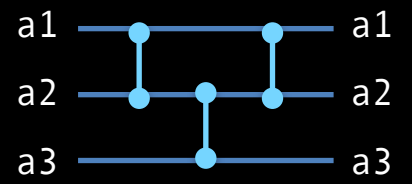
To sort n numbers, this will need n! checks!

Example: Sorting

- Read in 3 numbers and print them in order

```
#include <iostream>
using std::cout; using std::cin; using std::endl; using std::swap;
int main() {
  int a1, a2, a3;
  cin >> a1 >> a2 >> a3;
  if (a1 > a2)
      swap(a1,a2);
  if (a2 > a3)
      swap(a2,a3);
  if (a1 > a2)
      swap(a1,a2);
  cout << a1 << ", " << a2 << ", " << a3 << endl;
}
```

Sorting Network:



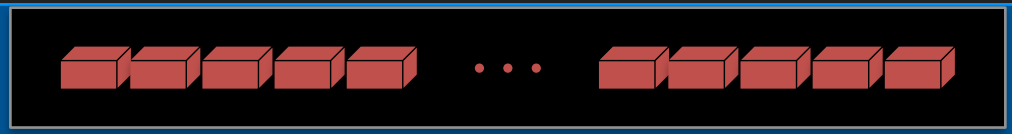
Much better!

But to sort a 100 numbers, will need thousands of lines of code (and quite error prone)!

Arrays to the Rescue!

Demo

- Arrays let us declare and use a large number of variables (of the same type)
- Declaration syntax: *type array_name* [*number*] ;
- Each variable in the array is referred to using an index: e.g., `N[i]`
  - The first element is `N[0]`, and the last element is `N[`*number*`-1]`



```
int N[26];
for (int i=0; i<26; i++)
  N[i] = 0;
char c;
for(cin >> c; c >= 'a' && c <= 'z'; cin >> c)
  N[c-'a'] ++;
for (char c='a'; c <= 'z'; c++)
  cout << "#" << c << " = " << N[c-'a'] << endl;
```

Index can be any integer valued expression

Example: Counting Letters

Accessing Array Elements

- Setting values during initialisation:
  - Explicitly: `int N[3] = {-1,0,1};`
  - Can initialise a few elements, and have the rest all set to 0
    - `int N[26] = {1};  // N[0]=1 and N[i]=0 for i=1 to 25`
    - `int N[26] = {};  // all values set to 0`
- At any point (not just initially), can assign values one by one: e.g., `N[i]=i;`
- There are a few other mechanisms for assigning values in bulk (involving concepts covered later). E.g., `std::fill` and `std::copy`
- Cannot assign to an array variable itself
  - ```
    int N[26];
    N = N; //error: array type 'int[26]' is not assignable
    ```
    ```

Example: Counting Vowels

Demo

- To read in a word (lower case) and count the occurrences of each vowel

```
#include <iostream>
using std::cout; using std::cin; using std::endl;
int main() {
  char c, vowels[] = {'a', 'e', 'i', 'o', 'u'}; // initialising explicitly
  int N[5] = {}; // initialising all elements to 0
  for(cin >> c; c >= 'a' && c <= 'z'; cin >> c)
    for (int i=0; i<5; i++)
      if(c == vowels[i]) {
        N[i]++; break;
      }
  for (int i=0; i<5; i++)
    cout << "#" << vowels[i] << " = " << N[i] << (i==4?"":", ");
  cout << endl;
}
```

Array Bounds

- It is the programmer's responsibility (rather than the compiler's) to ensure that when accessing an array, the index remains within bounds (i.e., 0 ≤ index ≤ number of elements -1)
- E.g., the compiler will not complain/warn about this:

```
int X[10], n;  cin  >> n;  X[n] = 1;
```

- Instead include a bound-check: `if (n>=0 && n<10)  X[n]=1; else …`
  - Or enforce index bounds in the program logic (e.g., `X[abs(n)%10]`)
- If the array index is out of bounds, it can cause the program to behave in unspecified ways (possibly crash, or access other variables)

Example: Palindrome using Arrays

- Read in an n-letter word and check if it is a palindrome

```
int main() {
  const int Nmax = 100; // array size needs to be known at compile time
  char text[Nmax];
  int n; cin >> n;
  if (n>Nmax) {
      cerr << "Text too long!" << endl; return -1;
  }
  for (int i=0; i<n; i++)  cin >> text[i]; // read it all into the array
  for (int i=0; i<n/2; i++) {
    if (text[i] != text[n-1-i]) {
      cout << "Not a palindrome!" << endl;  return 1;
    }
  }
  cout << "Palindrome!" << endl;
}
```

Many compilers (including g++) allow Variable Length Arrays (VLA) by default. But it is not part of the C++ standard. Avoid for portability!

Example: Palindrome using Arrays

Demo

- Read in an n-letter word and check if it is a palindrome

```
int main() {
  const int Nmax = 100; // array size needs to be known at compile time
  char text[Nmax], tmp;
  int n; cin >> n;
  if (n>2*Nmax + 1) {
      cerr << "Text too long!" << endl; return -1;
  }
  for (int i=0; i<n/2; i++)  cin >> text[i]; // read half into the array
  if (n%2) cin >> tmp; // if n odd, ignore the middle character
  for (int i=n/2-1; i>=0; i--) { // compare stored characters with rest
    if (cin >> tmp, text[i] != tmp) {
      cout << "Not a palindrome!" << endl;  return 1;
    }
  }
  cout << "Palindrome!" << endl;
}
```

Array Type: A Second Look

- Array declaration syntax is somewhat unusual
- `int A[10];` declares a variable called A of type `"int[10]"`
- But the following is invalid syntax: `int[10] A;  // wrong syntax`
- However the type `int[10]` can be given a new name, and it can be used instead

```
typedef int ints_10 [10]; // now ints_10 stands for int[10]
ints_10 A; // this has the same effect as int A[10];
```

- Can have a reference to an array: `ints_10& Z = A;`
  - Syntax without the typedef: `int (& Z) [10] = A;`
- But cannot have an array of references: `int& A[10]; // invalid`

Arrays and Functions

- A function can take array parameters

```
void readArray(int A[], int n) {
  for (int i=0; i<n; i++) cin >> A[i];
}
```

- Note: the array does not encode its size; needs to be passed separately
- Array elements are always passed by reference
  - Above, modifications to `A[i]` are reflected in the calling function
- A function cannot return an array type

Example: Sorting

Demo

- Read in n numbers and print them in order

```
#include <iostream>
using std::cout; using std::cin; using std::endl; using std::swap;
void max_to_end(int A[], int n) { // swap max(A[0],..,A[n-1]) with A[n-1]
    int maxi=0;
    for(int i=1; i<n; i++)
        if(A[i] > A[maxi])
            maxi = i;
    swap(A[maxi],A[n-1]);
}
int main() {
  ... // read numbers to be sorted into A[0] to A[n-1]
  for (int i=n; i>0; i--)
    max_to_end(A,i); // call max_to_end on the unsorted part of A
  ... // print out sorted A
}
```

Some Nuances

- Syntax allows a function's array parameter to mention a size, but it is not enforced when called!
  - E.g., `void f(int A[10]); int main(){ int X[2]; f(X); }` compiles without errors or warnings!
- References to arrays are allowed
  - As function parameters and as the return type
- Can use references to enforce array size of parameters.
  - E.g., `void f(ints_10& A);int main(){int X[2]; f(X);} // error`

```
typedef int ints_10 [10];
ints_10& g(ints_10& A, ints_10& B) {
  return A[0]>B[0] ? A : B;
}
int main() {
  ints_10& Z = g(X,Y);
}
```

Alternatives

- Arrays are somewhat restrictive data types
  - Size of an array must be known at compile time
    - Even when Variable Length Arrays are supported, it cannot be changed after the point of declaration
  - Cannot have expressions which are arrays
    - In particular, cannot pass an array by value to a function, or have a function return an array by value
- C++ standard library provides programmer-defined data types which are more flexible
  - `std::array` (allows passing/returning by value, simply by wrapping in a struct) and `std::vector` (also allows resizing, and more). **Later.**