CS 101: Computer Programming and Utilization

09-Arrays

Instructor: Sridhar Iyer
IIT Bombay

Printing fibonacci numbers

Recall from lab04: fibonacci numbers

- The Scratch program that you wrote.
- The C++ program that you executed.
- These programs output each fibonacci number as soon as it is computed.

Now, modify your program to first generate and store the fibonacci numbers somewhere, and then output them.

Run: demo09-fibonacciArray.sb; demo09-fibonacciArray.cpp

Array (1-dimensional)

- So far, each `int`, `double` or `char` variable had a distinct name
  - Called “scalar” variables
- An array is a **collection** of variables
  - They share the same name
  - But they have different **indices** in the array
- Arrays are provided in C++ in two ways
  - “Native arrays” supported by language itself
  - The `vector` type (will introduce later)
- Similar: understand one, understand both

Notation

- When writing series expressions in math, we use subscripts like a_i, b_j etc.
- For writing code, the subscript is placed in box brackets, as in a[i], b[j]
- Inside […] can be an arbitrary integer expression

- In C++, the first element is a[0], b[0] etc.
- If the array has n elements, the last element is at position n-1, as in a[n-1]
- Watch against out-of-bound index errors

Why do we need arrays: Another example

- Print a running median of the last 1000 temperatures recorded at a sensor
- Tedious to declare and use 1000 scalar variables of the form t0, t1, …, t99
- Cannot easily express a computation like “print differences between consecutive readings” as a loop computation
- Want to write “$t_{i+1} - t_i$”
- i.e., want to access elements using index that is an integer expression

Pair-Activity: What does this program do?

```
main() {
  int vn = 9;
  int va[vn];
  for (int vx = 0; vx < vn; vx++) {
    va[vx] = vx  * (vn - 1 - vx);
  }

  for (int vx = 0; vx < vn; vx++) {
    cout << va[vx] << ", ";
}
cout << endl;
```

Number of elements in array to be created

Reserve memory for 9-element array

Lvalue: cell to which rhs value is to be written

Rvalue: access int in specified cell

Activity: Find min and max of array

Recall the pair-activity – demo09-array.cpp program to initialize an array and print it.

Now you have to extend this program to find the smallest and largest element in the array.

Think: Write the pseudo-code individually.

Pair: Write the c++ code with a partner.

Share: Compare with demo09-array-mod1.cpp

Extensions to previous program

Extension 1:

- Discuss: Initialize the array with random numbers.
  - See demo09-array-mod2.cpp

Extension 2:

- Discuss: Extend *array*.cpp to find index of smallest and index of largest elements.
  - See demo09-array-mod3.cpp

Extension 3: See next slide

Activity: Exchange Sort

Extension 3:

Exchange the smallest element with a[0].

Repeat appropriately till the array is sorted.

- Think: Write the psuedo-code for the above extension.
- Pair: Write the c++ code with a partner.
- Share: Compare demo09-array-mod4.cpp

Example: Sum and product of all elements

```
double arr[an]; // fill in suitably
double sum = 0, prod = 1;
for (int ax = 0; ax < an; ++ax) {
  sum += arr[ax];
  prod *= arr[ax];
}
```

- In standard notation we would write these as

$$\sum_{0 \le i < n} a_i \qquad \prod_{0 \le i < n} a_i$$

Example: Dot-product of two vectors

```
double av[nn], bv[nn]; // filled in
double dotprod = 0;
for (int ix = 0; ix < nn; ++ix) {
  dotprod += av[ix] * bv[ix];
}
```

$$\sum_{0 \leq i < n} a_i \, b_i$$

Self-study: Prefix (cumulative) sum

- Given array a[0,…,n-1]
- Compute array b[0,…,n-1] where b[i] = a[0] + a[1] + … + a[i]

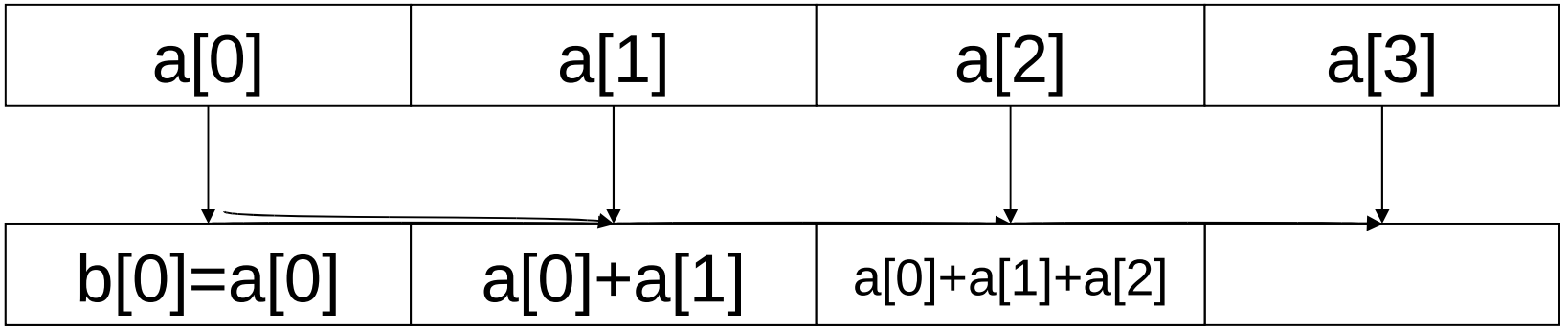
Nested loop below works; can you improve it?

```
int a[n], b[n]; // array a is given filled
for (int bx = 0; bx < n; ++bx) {
  b[bx] = 0;
  for (int ax = 0; ax <= bx; ++ax) {
    b[bx] += a[ax];
  }
}
```

Time taken is proportional to n^2

Prefix sum – faster computation

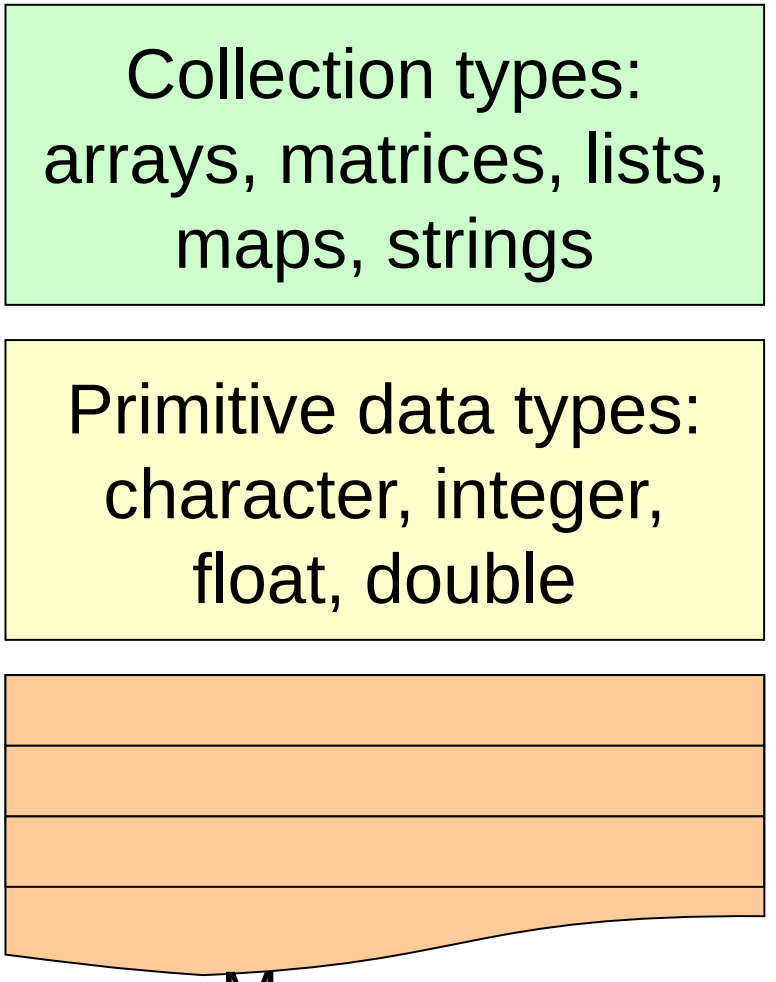
- Given array a[0,…,n-1]
- Compute array b[0,…,n-1] where
  b[i] = a[0] + a[1] + … + a[i]




```
int a[n], b[n];
// vector a filled
for (int ix = 0; ix < n; ++ix) {
  b[ix] = ( (ix == 0)? 0 : b[ix-1]) + a[ix];
}
```

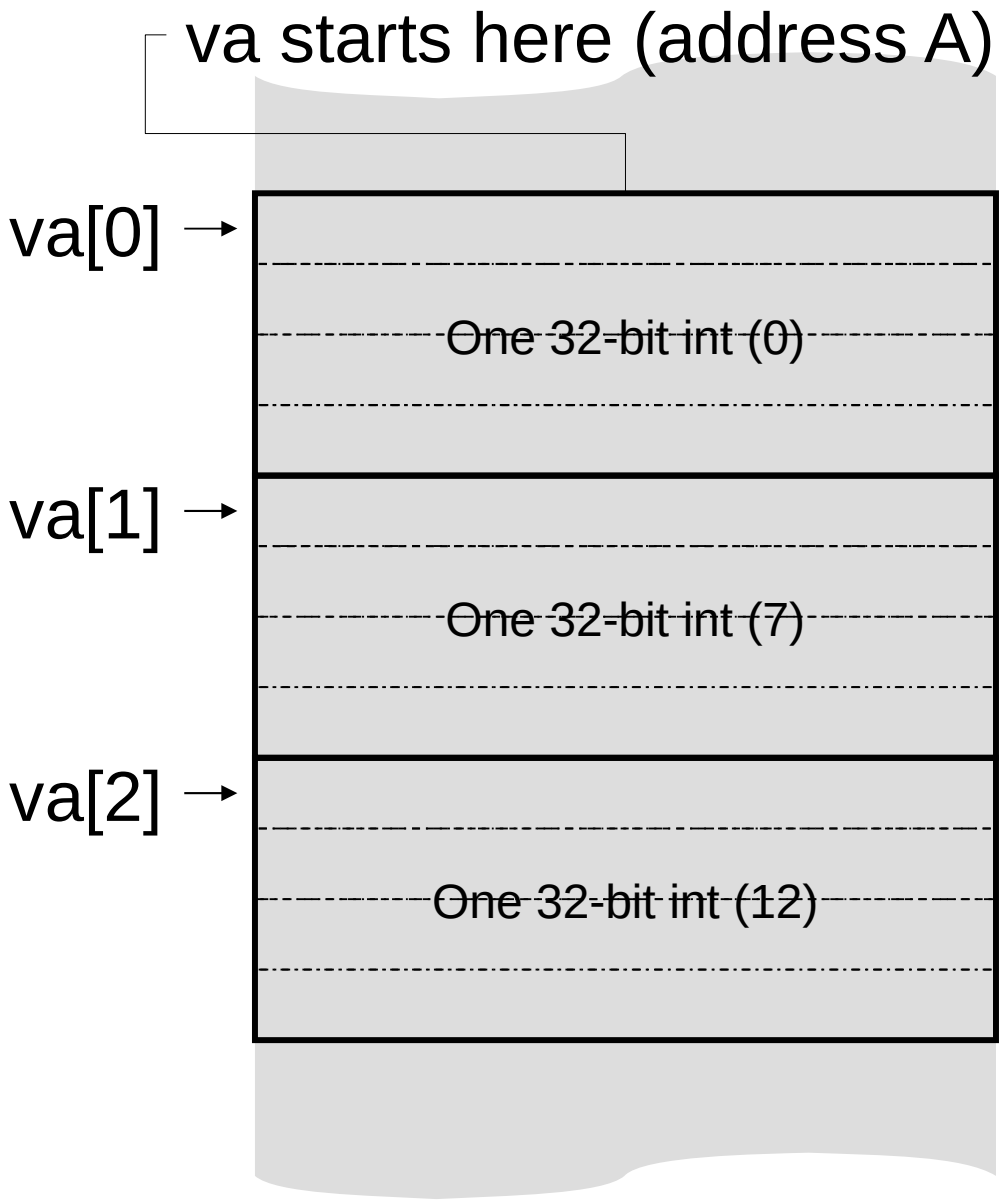
Optional: More on data types

- Three layers of abstraction
- Implement fixed size primitive types by mapping possible/ supported values to bit patterns
- Add collection types on top of primitive types to assist writing complicated programs
- Collections can change sizes and memory layout during program execution



Optional: Array - Representation in memory

- Elements of array have fixed size
- Laid out consecutively
- Compiler associates array name va with memory address of first byte
- To fetch va[ix], go to byte address A + ix*4 and fetch next four bytes as integer
- A, A+4, A+8, A+12, …



Reading Notes

- See slides from session08 of Prof. Phatak's cs101-2011 course:
  - http://www.cse.iitb.ac.in/~cs101/2011.2/Lectures/lecture_slides/2011_08_19_session08_arrays.pdf