APS105 Lecture 14
6.4, 6.3

Dan Zingaro

February 26, 2010

Feedback from Reading Quiz

- "I found 3d arrays difficult to visualize"
  - So do some of your instructors. I can't remember the last time I used one
- "I dont understand `malloc` ..."
  - I get this one on every reading quiz ... please don't think about `malloc` yet
- Using `*` instead of `[]`, give an equivalent way of writing `a[i]`, where `a` is a one-dimensional array, and `i` is an integer.
  - So, we have to use `*` to get element `i` of the array
  - Using `a` by itself gives us the address of its first element
  - Using `a + i` gives us the address of element `i`
  - To get the value at address `a + i`: `*(a + i)`

Maximum Segment Sum

- The problem we'd like to solve is the **maximum segment sum**
- Input: array `s` of `n` integers
- Output: maximum sum of any segment of `s`
- A segment is a contiguous portion of the array
- For example, the segments of the array `[2, -4, 6]` are `[]`, `[2]`, `[-4]`, `[6]`, `[2, -4]`, `[-4, 6]`, `[2, -4, 6]`
- Question: what is the maximum segment sum here?
- Question: is `[2,6]` a segment of `[2, -4, 6]`?

ConcepTest

Choose an array that has the largest maximum segment sum.

- A. `[4, -3, 9, -5]`
- B. `[-5, 10, -6, 2]`
- C. `[-3, -2, -1]` (careful!)

Segment Observations

- The maximum segment sum of an array of all positive numbers is the sum of all of the numbers
- When we have some negative numbers in the array, it gets trickier. Which ones do we include?
- If all numbers are negative, the maximum sum is 0, realized by the empty segment
- We can list every segment of a list by pairing each possible starting point with each possible ending point

Summing each Segment

- Let's write a program that computes the sum of each possible segment in the array
- For example, here is how we will start operating on `[4, -3, 9, -5]`

```
max: 0
sum of segment [4] = 4
max: 4
sum of segment [4,-3] = 4-3 = 1
max: 4
sum of segment [4,-3,9] = 4-3+9 = 10
max: 10
sum of segment [4,-3,9,-5] = 4-3+9-5 = 5
max: 10
... continue with sum of segment [-3] ...
```

Two-dimensional Arrays

- Assume we have three students, each with four marks

	Mark 0	Mark 1	Mark 2	Mark 3
Student 0	85	82	89	44
Student 1	51	52	98	99
Student 2	50	60	70	80

- This is a table with three rows and four columns
- We can declare a two-dimensional array of this shape

Two-dimensional Arrays...

```
int marks[3][4];
```

- The first number (3) is the number of rows
- The second number (4) is the number of columns
- We then access individual elements by indexing twice
- e.g. the second student's first mark can be set with `marks[1][0] = 51`
- To access every element of the array, we require nested loops

```
for (int i = 0; i < 3; i++)
  for (int j = 0; j < 4; j++)
    ... do something with marks[i][j]
```

ConcepTest

What is the proper loop structure for accessing the elements in the following table **by column**, from top to bottom? We want to access the elements in the first column, then those in the second column, etc.

```
int marks[3][4];
```

- A.

```
for (int i = 0; i < 4; i++)
  for (int j = 0; j < 3; j++)
  ... access marks[i][j]
```

- B.

```
for (int i = 0; i < 3; i++)
  for (int j = 0; j < 4; j++)
    ... access marks[i][j]
```

- C.

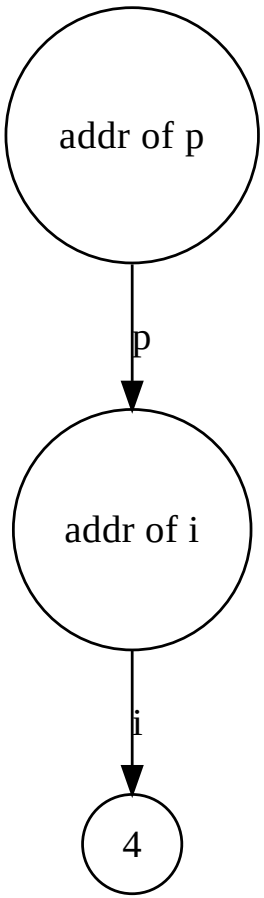
```
for (int i = 0; i < 4; i++)
  for (int j = 0; j < 3; j++)
    ... access marks[j][i]
```

Pointers

- Each variable in a program occupies one or more bytes of memory
- A variable's address (location) is the address of its first byte
- We use pointer variables to store addresses
- e.g. A pointer variable that can hold the address of where an `int` can be found: `int *p;`
- To get the address of a variable: `&`
- To get whatever is stored at the address in a pointer variable: `*`
  - We should only do this if we have already initialized a pointer to point to a value

Pointers: Example

```
int i = 4;
int *p;
p = &i;
```

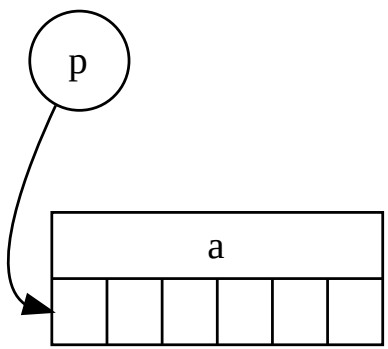


Pointers and Arrays

- Since array elements are just variables, pointers can point to array elements

```
int a[5], *p;
p = &a[0];
```

- Now, we can access or change `a[0]` through `p`
- e.g. `*p = 5;`



Pointer Arithmetic

- If `p` points to an element of an array, `p+j` points `j` elements to the right
- If `p` points to an element of an array, `p-j` points `j` elements to the left

```
int a[10], *p, *q;
p = &a[2];
q = p+3; // q points to a[5]
*q = 84; //a[5] now = 84
```

ConcepTest

What are the value of array a's elements after the following code?

```
int a[2] = {2, 4};
int *p = &a[1];
*p = 1;
p--;
*(p+1) = 6;
```

- A. [2, 4]
- B. [1, 6]
- C. [6, 1]
- D. [2, 6]
- E. [6, 4]