Arrays in C

Vector Defined

- Ordered List of Values
- All of the same type
- Individual elements accessible by “index”
- Vector has a Size (Number of elements)

Why Use a Vector?

- When your data is a list of values
- For instance, suppose you have a list of vertices in a rectangle

```
float vx[4];
float vy[4];
vx[0]=  0.0; vy[0]=0.0;
vx[1]=10.0; vy[1]=0.0;
vx[2]=10.0; vx[2]=4.0;
vx[3]=  0.0; vy[3]=4.0;
```

Vector Declaration

<type> <name>[<size>];

- <type> - Any built-in or derived data type
  - int, char, short, float, etc.
- <name> - Any valid variable name
  - e.g. vx, vy, myArray, etc. etc.
- <size> - Integer constant - how many items are in the list

- Vectors must be declared before they are used.

Vector Size

- Number of elements for each dimension
- int vec[12]; // Reserve space for 12 integers
- WARNING: Indexes are 0,1,2,3,…, 9, 10, 11!

Referencing Vector Values

<name>[<index>]

- <name> - The (declared) name of a vector
- <index> - The index of a specific element in the vector
  - STARTS AT ZERO!!!!!
  - Maximum value is (Size – 1)

Example Vector Code

```
int grades[14];
…
int j,sum=0;
for(j=0; j<14; j++) {
        sum +=grades[j];
}
float avg=(float)sum/14;
printf("Average grade: %f\n",avg);
```

Matrix – Two Dimensional Array

- Declaration: <type> <name>[<rows>][<cols>];

- Reference: <name>[<row_index>][<col_index>]
  - 0 <= row_index < rows
  - 0 <= col_index < cols

Why Use a Matrix?

- When your data is rectangular in nature

- For example, Grades for Multiple Students
  - Each student takes one row
  - Each grades takes one column

Example Matrix Code

```
int grades[20][14];
…
int st;
for(st=0;st<20;st++) {
        int gr,sum=0;
        for(gr=0;gr<14;gr++) sum+=grades[st][gr];
        printf("Average for student %2d: %f\n",st,sum/14.0);
}
```

Array Dimensions

Vector: int vec[4]={10,20,30,40};

vec[0]	vec[1]	vec[2]	vec[3]
10	20	30	40

Matrix: int matrix[2][3]={10,11,12,20,21,22}

matrix[0][0] 10	matrix[0][1] 11	matrix[0][2] 12
matrix[1][0] 20	matrix[1][1] 21	matrix[1][2] 22

Cube: char cube[3][2][3] = { “abcdefghijklmnopqr”};

[0][0][0] ‘a’	[0][0][1] ‘b’	[0][0][2] ‘c’
[0][1][0] ‘d’	[0][1][1] ‘e’	[0][1][2] ‘f’

[1][0][0] ‘g’	[1][0][1] ‘h’	[1][0][2] ‘i’
[1][1][0] ‘j’	[1][1][1] ‘k’	[1][1][2] ‘l’

[2][0][0] ‘m’	[2][0][1] ‘n’	[2][0][2] ‘o’
[2][1][0] ‘p’	[2][1][1] ‘q’	[2][1][2] ‘r’

Array Values are "Contiguous"

- Right next to each other in memory
- int vec[6]

vec[0]	vec[1]	vec[2]	vec[3]	vec[4]	vec[5]

- int m [4][3];

m[0][0]	m[0][1]	m[0][2]	m[1][0]	m[1][1]	m[1][2]	m[2][0]	m[2][1]	m[2][2]	m[3][0]	m[3][1]	m[3][2]

Array Bounds Checking

```
int vec[5];
for(i=0;i<=5;i++) vec[i]=12;
```

vec[0] 12	vec[1] 12	vec[2] 12	vec[3] 12	vec[4] 12	??????? 12

- NO RUN-TIME ARRAY BOUNDS CHECKING IN C!!!!!!!!!!!
- Trust the programmer, and save the run-time!
- Programmer must be trustworthy!
- “vec[5]=12;” causes compile error… not run-time.

"Row Major Order"

- Think of multi-dimensional indexes as an odometer…
- Rightmost digit of index increases the fastest
- Once rightmost digit reaches it's limit, it goes back to zero, and
- Digit to the left increases by 1

Array Initialization

<type> <name>[<size>] = { <list_of_constants> }

- Each constant separated by a comma
- For multi-dimensional arrays, initialization is in row major order
- If list is too short, padded with zeroes

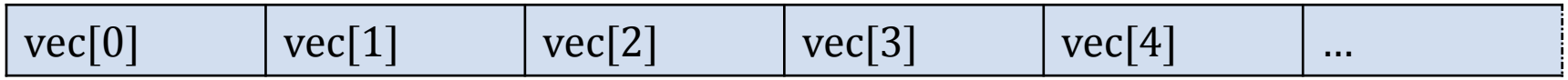
- ***IF ARRAY IS NOT INITIALIZED IT'S INITIAL VALUE IS UNKNOWN!***
  - Whatever value was in memory when the functions starts

Variable Size Arrays

- C allows first dimension to have unspecified size

```
int vec[];
```

  - vec is an list of some number of integers…



  - Programmer must know how many elements are usable!

```
int mat[][4];
```

  - mat is an array in which each row has 4 integers
  - Programmer must know how many rows are usable

Resources

- Programming in C, Chapter 6

Quiz 2

1. In gdb, is the command “p 3*x” a valid command?

2. If the current value of x is 12, what will the command “p x+2” print?

3. True or False: The invocation record for main holds the local variables for the main function.

4. True or False: The stack of invocation records never has more than two invocation records for a recursive function.