iClicker Attendance

Please click on A if you are here:

A. I am here today.

Review Demo : Conversion

Given a floating point number, return an integer representation of that number rounded using various different rounding policies:

- round up
- round down
- round towards zero
- round towards +/- infinity
- round to the nearest integer

… Without using C library functions

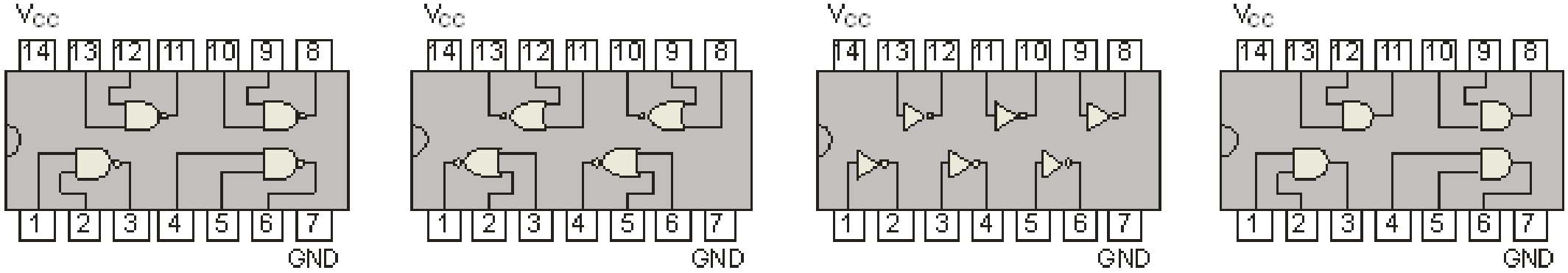
Arrays in C

One Dimensional Array (Vector)

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

Index	0	1	2	3	4	5
Value	17.3	14.5	3.2	12.0	5.65	14.5

Why Use a Vector?



- When your data is a list of values
- For instance, gates per chip

```
int gpc[4];
gpc[0]=gpc[1]=gpc[3]=4; gpc[2]=6;
```

0	1	2	3
4	4	6	4

One Dimensional Array 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.
- Once the size is specified, it cannot be changed!

Vector Size

- Number of elements

```
int vec[4]; // Reserve space for 4 integers
```

- WARNING: Indexes are 0,1,2,3
  - Index starts at 0
  - Index ends at (SIZE-1)
- To find the size of an array: sizeof(vec)/sizeof(int)
  - sizeof calculates the number of bytes needed for a variable or type
  - In the above example, sizeof(vec)=36, sizeof(int)=4

Referencing Vector Values

name[*index*]

- *name* - The (declared) name of a vector
- *index* - The index of a specific element in the vector
  - "First" element in the vector is `vec[0]`
  - "Last" element in the vector is `vec[size-1]`
- Index may be any valid integer expression
  - e.g. : `vec[3]`, `vec[j]`, `vec[2*i+1]`

Example Vector Code

```
int grades[14];
… // Code which fills in grades goes here
int j,sum=0;
for(j=0; j<14; j++) {
        sum +=grades[j];
}
float avg=(float)sum/14.0;
printf("Average grade: %f\n",avg);
```

Initializing a Vector

```
int x=7; // Initializing a scalar variable
int gpc[4]={4,4,6,4}; // Initializing vector
```

0	1	2	3
4	4	6	4

- Or, let the compiler count the number of elements

```
int gpc[ ]= {4,4,6,4};
```

Array Declaration with Initialization

type name[*size*] = { *list_of_constants* };

- Each constant separated by a comma
- If size specified, and list is too short, padded to the right with zeroes
- If size is not specified, size is number of elements in list

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

C Idiom: Zeroing out an Array

- Initialization “rule”… if you specify an initializer that does not contain enough values, C will “pad” your initializer to the right with 0.

- Thus, to initialize an entire array to zero, just specify a single 0 value.

```
int countOranges[37] = { 0 }; // All 37 orange boxes are empty
```

iClicker Question

- What gets printed?

```
int vec[5]; int i;
for(i=0;i<=5;i++) vec[i]=4;
printf("vec[3]=%d\n",vec[3]);
```

A. No value – compiler error.
B. Nothing – endless loop
C. vec[3]=4
D. Segmentation violation

Pitfall: Array Bounds Checking

```
int vec[5]; int i;
for(i=0;i<=5;i++) vec[i]=4;
```

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

- NO RUN-TIME ARRAY BOUNDS CHECKING IN C!!!!!!!!!!!
- Trust the programmer, and save the run-time!
- Programmer must be trustworthy!
- Writing past the end of an array can cause many problems
  - May write over other variables
  - May cause a segmentation violation

Matrix – Two Dimensional Array

- Declaration: *type name*[*rows*][*cols*];

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

- Indexes may be any integer expressions

Why Use a Matrix?

- When your data is rectangular in nature

Student	Quiz1	Homework1	Test1	Homework2	…
John Smith	98	76	68	82	…
Alice Jones	73	94	86	79	…
Jane Jameson	62	73	68	70	…
…	…	…	…	…	

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

Example Matrix Code

```
int grades[20][14]; // Space for 20 students, 14 grades
…
int st;
for(st=0;st<20;st++) {
        int gr; int 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

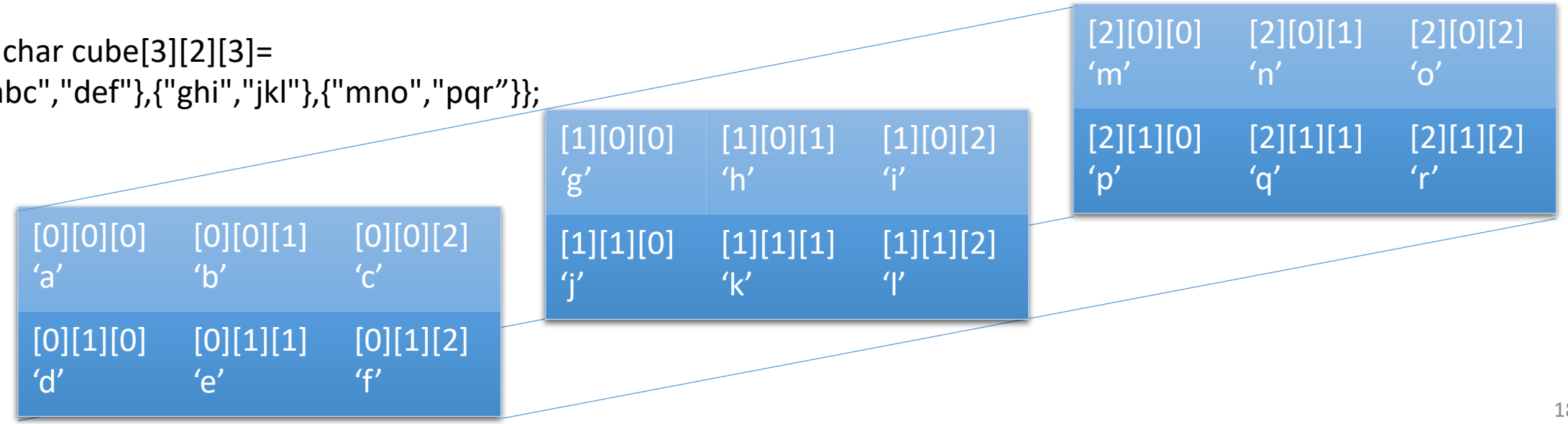
Vector: int vec[4]={10,20,30,40};

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]=
{{"abc","def"},{"ghi","jkl"},{"mno","pqr"}};



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]

- If we know where the first element is, we know the entire array!

"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

```
int m[4][3]={0,1,2,10,11,12,
             20,21,22,30,31,32};
```

	0	1	2
0	0	1	2
1	10	11	12
2	20	21	22
3	30	31	32

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]
0	1	2	10	11	12	20	21	22	30	31	32

Using Entire Arrays

- We know how to manage individual elements of an array
  - Individual elements are primitive types – numbers, characters, etc.

```
int gpc[4]={4,4,6,4}; // Initializing vector
printf("The third chip has %d gates\n",gpc[2]);
```

- What can we do with an ENTIRE array?

```
printf("The gpc array is %x\n",gpc);
```

Array "Types"

- "Type" tells the compiler how to interpret a value
  - int, float, char, etc.
- An array has a "type" – tells the compiler how to read the array
  - Includes the number of dimensions
  - Includes the size of each dimension
  - Includes the type of each element
- For instance, to declare a parameter which is a vector:

```
int sum(int grades[17]) { …
```

Array Parameters with Size

- See arrayArg_simple.c

- Declare a parameter with a fixed size
- C Assumes the argument is an array of the given size!
  - Compiler does not check
  - If array is actually smaller, no warning/error messages – just wrong results

Array Parameters are REFERENCES

- The argument is not actually an array… it is a REFERENCE TO an array
- The parameter (inside the function) is a COPY OF the REFERENCE!

- If the function modifies the array it is modifying the ACTUAL array and your caller can see those modifications!

- See arrayArg_mod.c

Array Parameters with Undefined Size

- C allows specification of an array parameter without a specific size
- The function must still know what the correct (valid) size should be
  - For instance, with a separate "size" parameter
- For multi-dimensional arrays, only the FIRST size can be unspecified
- See arrayArg_usize.c

Using "Guard" values

- One strategy is to define a value that is "invalid" in the array
- For instance, in gates per chip, -1 is an invalid value
- Use the invalid value to identify the size of the array
  - Everything up to, but not including the guard value is valid

- See arrayArg_guard.c

C "strings"

- Strings are an array of characters with a guard value of 0x00
  - 0x00 is called a "null terminator"
  - There is no ASCII printable character with the value 0x00
- Note: in C, "char string[]" and "char * string" mean exactly the same thing – an array characters of unknown length (hopefully with a null terminator).
- Warning: char * string does not allocate memory for a string
  - char string[]="This is a string"; does

String Literals

- The literal "my string" is the same as… {'m','y',' ','s','t','r','i','n','g',0x00}
  - An array initializer with character elements
  - Note that this is the same as…

    {0x6d, 0x79, 0x20, 0x73, 0x74, 0x72, 0x69, 0x6e, 0x67, 0x00 }

```
char str[]="my string";
printf("string length of \"%s\" is %d, length of str is %d\n",
        str,strlen(str),sizeof(str));
```

prints: string length of "my string" is 9, length of str is 10

Demo

Make an array of characters from the common phrase “we hold these truths to be self evident; that all men are created equal”.

Find and print the index of every ‘e’ in this phrase

Resources

- Programming in C, Chapter 6 (Arrays)
- C FAQ Arrays and Pointers: http://c-faq.com/aryptr/index.html
- C Tutorial on Arrays: http://www.crasseux.com/books/ctutorial/Arrays.html#Arrays
- Wikipedia Array Data Structure: https://en.wikipedia.org/wiki/Array_data_structure
- Wikipedia C Arrays section: https://en.wikipedia.org/wiki/C_(programming_language)#Arrays