Cat & Mouse

```
input catRadius, catAngle, mouseAngle
for minute = 1 to 30
    if cat catches mouse in next move
        break
    if cat sees mouse
        move cat in
    else
        move cat around statue
    move mouse
    reduce angles to < 2*pi
    output positions
end
print message
```

Cat catches mouse in next move

```
if catRadius != mouseRadius
    return no
move cat around statue
if(newCatAngle == mouseAngle ||
angle between two others)
    return yes
```

Arrays

CS2023 Winter 2004

Outcomes: Arrays

- “C for Java Programmers”, Chapter 10
- Other textbooks on C on reserve
- After the conclusion of this section you should be able to
  - Declare, initialize, and traverse C arrays
  - Pass arrays to functions
  - Start working with multidimensional arrays

One-dimensional Arrays

- C arrays:
  - have a lower bound equal to zero
  - are *static* - their size must be known at compile time.
- To define an array:

```
type arrayName[size];
```

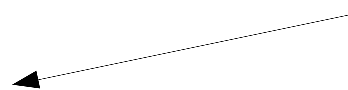
For example

```
int id[1000];
#define SIZE 10
double scores[SIZE+1];
```

One-dimensional Arrays

It is good programming practice to define size of an array with a macro

```
#define N 100
...
int a[N];
for (i=0; i < N; i++)
   sum += a[i];
```



Idiom!

Avoids error of falling off end of array

Array Initialization

- Most common form:

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- If list of initializers is shorter, remaining elements initialized to zero.

```
int a[10] = {1, 2, 3, 4, 5, 6};
```

- Can omit length of array when initializing

```
int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Array Subscripting

- **a[*expr*]**, when ***expr*** is an integer expression
- **a[*expr*]** can be used in same way as ordinary variables

```
a[0] = 1;
printf("%d\n", a[5]);
++a[i];
```

- Other idioms with for loops

```
for (i=0; i < N; i++)
  a[i] =0;              /* clears a */

for (i=0; i < N; i++)
  scanf("%d", &a[i]); /* reads data into a */
```

Array Subscripting

- C doesn't require that subscript bounds be checked!

```
int a[N],i;
for (i=0; i <= N; i++)
   a[i] =0;
```

Will compile and run, but could cause infinite loop!

```
/* Checks numbers for repeated digits */
#define TRUE 1
#define FALSE 0
typedef int Bool;

int main()
{
  Bool digit_seen[10] = {0};
  int digit;
  long int n;
  printf("Enter a number: ");
  if(scanf("%ld", &n) != 1)
    return 1; /* should also print error message */
  while (n > 0) {
    digit = n % 10;
    if(digit_seen[digit])
      break;
    digit_seen[digit] = TRUE;
    n /= 10;
  }
  printf("%s repeated digits\n", n > 0 ? "" : "No");
  return 0;
}
```

Using `sizeof` with Arrays

sizeof gives number of bytes of argument:

```
int a[20];

sizeof(a); \* 20*sizeof(int) *\

sizeof(a) / sizeof(a[0]); \* 20 *\


#define SIZE(x) sizeof(x) / sizeof((x)[0])

for (i = 0; i < SIZE(a); i++)
  a[i] = 0;
```

Array errors

```
int arrayName[SIZE];
arrayName[SIZE] = 2;
```

```
int n = 3;
double s[n];
```
size of array must be constant

```
s = 5;
```
Arrays are not l-values

Side-effects make the result of assignments involving index expressions *implementation dependent*

```
a[i] = i++;
```

Array Arguments in Functions

- Length of array is left unspecified:

```
int f(int a[])
{...}
```

- If length is needed, must be specified with extra parameter:

```
int f(int a[], int n)
{...}
```

- **Can't use `sizeof` to determine length of an array parameter**, as we just did for array variable

```
int f(int a[]){
  int len = sizeof(a) / sizeof(a[0]) /* WRONG */
.. }
```

Array Arguments in Functions

- When calling a function with an array argument, simply pass the array name:

```
#define LEN 100
int f(int a[], int n);
int main()
{
  int g, a[LEN]
  ...
  g = f(a, LEN) /* g = f(a[], LEN) is WRONG */
  ...
}
```

```c
#define LEN 100

int main()
{
  int b[LEN], total;
  ...
  total = sum_array(b, LEN);
  ...
}


int sum_array(int a[], int n)
{
  int i, sum = 0;

  for (i = 0; i < n; i++)
    sum += a[i];

  return sum;
}
```

```
#define LEN 100

int main()
{
  int b[LEN], total;
  ...
  total = sum_array(b, 50); /* sum first 50 */
/* total = sum_array(b, 150) is WRONG! */
  ...
}

int sum_array(int a[], int n)
{
  int i, sum = 0;

  for (i = 0; i < n; i++)
    sum += a[i];

  return sum;
}
```

Array Arguments in Functions

- When array passed as argument to a function, base address is passed "call-by-value", and *not* the array elements themselves.
  - See pointers (soon!)
- This means arrays that are passed as parameters to functions can be modified by those functions

```c
#define LEN 100

int main()
{
  int b[LEN], total;
  ...
  read_array(b, LEN);
  total = sum_array(b, LEN);
  ...
}


void read_array(int a[], int n)
{
  int i, sum = 0;

  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
}
```

Multidimensional Arrays

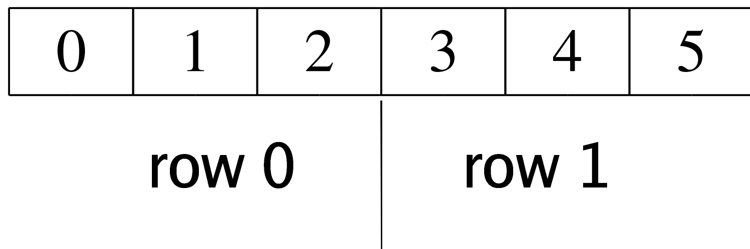
- An array can have any number of dimensions
- Create a two-dimensional array:

```
int m[2][3];
```

- To access element in row i and column j:

```
m[i][j]
```

- Arrays stored in row-major order:



Initializing Multidimensional Arrays

- Nest one-dimensional initializers:

```
int m[2][3] = {{1, 0, 1},{0, 1, 0}};
```

- Inner braces can be omitted

Multidimensional Arrays as Parameters

- Only length of first dimension may be omitted

```
/* call using j = sum_array(b, nrows) */
int sum_array(int a[][LEN], int n)
{
  int i, j, sum = 0;

  for (i = 0; i < n; i++)
    for (j = 0; j < LEN; j++)
      sum += a[i][j];
  return sum;
}
```

- Can work around this using pointers (soon!)