CSE 220 – C Programming

Arrays

Data Types

- Scalar Types: hold a single a value
  - float
  - int
  - char, …
- Aggregate Data Types: referenced as a single entity but hold several values
  - array
  - struct

One Dimensional Array

- Data structure containing a number of values of the same type
- To declare an array, specify:
  - Type of variables stored
  - Constant number: for the number of elements
  - Name of the array

```
int a[8];
```

Access

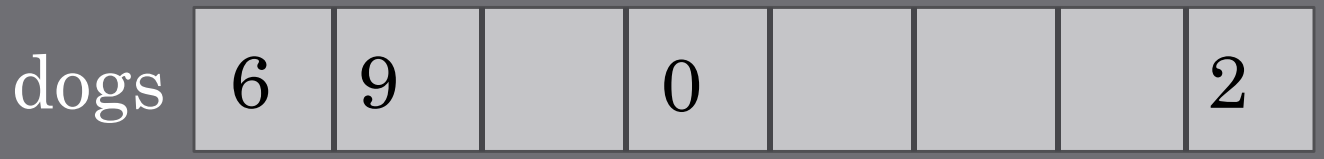
- Index starts at 0
- If size n, last index is n-1


```
//Set values
#define Z 1
a[0] = 11;
a[Z] = 20;
a[2+2] = 0;
```

```
//Print cell content
printf("%d\n", a[0]);
```

```
//Read into a cell
scanf("%d\n", &a[6]);
```

Exercise



With the array above, what would this line output:

```
printf("Josh has %d dog(s).", dogs[1]);
```

- Josh has %d dog(s).
- Josh has 6 dog(s).
- Josh has 9 dog(s).
- Josh has 2 dog(s).

Example

Read 10 integers from the user and print them backwards:

```
//Declare array
int values[10];
int idx;
//Read values
for (idx=0; idx<10; idx++) {
        scanf(“%d”, &values[idx]);
}
//Print from last to first
for (idx=9; idx>=0; idx--) {
        printf(“%d”, values[idx]);
}
```

What changes need to be done if we need an array of 11 elements?

Example

Read 10 integers from the user and print them backwards:

```
//Declare array
int values[11];
int idx;
//Read values
for (idx=0; idx<11; idx++) {
        scanf("%d", &values[idx]);
}
//Print from last to first
for (idx=11-1; idx>=0; idx--) {
        printf("%d", values[idx]);
}
```

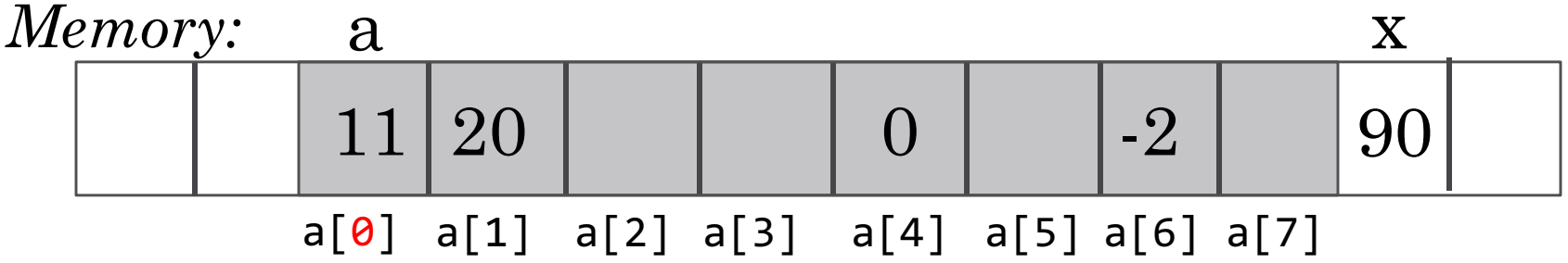
What changes need to be done if we need an array of 11 elements?

Example

```
#define  size 10
…
int idx;
int values[size];
//Read values
for (idx=0; idx<size; idx++) {
        scanf(“%d”, &values[idx]);
}
//Print from last to first
for (idx=size – 1; idx>=0; idx--) {
        printf(“%d”, values[idx]);
}
```

Bounds

- C does not check subscript bounds




```
int x = 7;
int a[8];
a[8] = 90;
printf("x is:%d\n", x);
```

Initialization

- Initialize to constant values:

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

char myLetters[5] = {‘a’, ‘z’, ‘x’, ‘1’, ‘w’};
```

- If initializer is shorter than array size, the remainder of the array is initialized to zero:

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

- If initializer is longer than array size: error
- If size omitted and initializer present, the array will have the initializer value:

```
int a[ ] = {1, 2, 3, 4, 5};
//Same as int a[5] = {1, 2, 3, 4, 5};
```

Exercise

Which of the following arrays accesses are out-of-bounds for the array `grades`:

```
int grades[] = {7, 4, 6, 6};
```

- `grades[-1]`
- `grades[0]`
- `grades[3]`
- `grades[4]`

Exercise

Which of the following arrays accesses are out-of-bounds for the array `grades`:

```
int grades[5] = {7, 4, 6, 6};
```

- `grades[-1]`
- `grades[0]`
- `grades[3]`
- `grades[4]`

Example

```
switch (month) {
        case 1: mc = 0; break;
        case 2: mc = 2; break;
        case 3: mc = 3; break;
        case 4: mc = 1; break;
        case 5: mc = 3; break;
        case 6: mc = 0; break;
        case 7: mc = 4; break;
      …
}
```

```
int mc[12] =
      {0,2,3,1,3,0,
      4,2,3,4,1,2};

//Use in formula as:
mc[month – 1]
```

Exercise

- Double the values in the given array:

```
int myValues[7] = {1, 4, 0, -1, 7, 9, 1};
for (int idx=0; idx < 7; idx++) {
     myValues[idx] = myValues[idx]*2;
}
```

Size of arrays

- sizeof operator:

```
int a[10] = {1, 2, 3, 4, 5,
                6, 7, 8, 9, 10};
int size_a = sizeof(a);
//10*4 (4 bytes for an int)
int size_int = sizeof(a[0]);
//4
int num_elements = sizeof(a) / sizeof(a[0]);
//10
```

Multidimensional Arrays

- Arrays may have any number of dimensions:

```
int temp[5][3]; //5 Rows, 3 Columns
```

temp		*0*	*1*	*2*
	0	temp[0][0]	temp[0][1]	temp[0][2]
	1	temp[1][0]	temp[1][1]	temp[1][2]
	2	temp[2][0]	temp[2][1]	temp[2][2]
	3	temp[3][0]	temp[3][1]	temp[3][2]
	4	temp[4][0]	temp[4][1]	temp[4][2]

- To access row i and col j: `temp[i][j]`

Multidimensional Arrays

- In memory:

temp

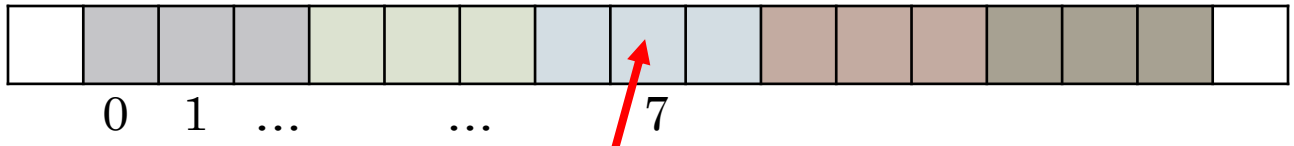
Logical Representation:

temp

temp[0][0]	temp[0][1]	temp[0][2]
temp[1][0]	temp[1][1]	temp[1][2]
temp[2][0]	temp[2][1]	temp[2][2]
temp[3][0]	temp[3][1]	temp[3][2]
temp[4][0]	temp[4][1]	temp[4][2]

Multidimensional Arrays

temp



temp

temp[0][0]	temp[0][1]	temp[0][2]
temp[1][0]	temp[1][1]	temp[1][2]
temp[2][0]	temp[2][1]	temp[2][2]
temp[3][0]	temp[3][1]	temp[3][2]
temp[4][0]	temp[4][1]	temp[4][2]

Set temp[2][1] to 45:

```
temp[2][1] = 45;
```

```
temp[7] = 45;
```

Solution

- Print the content of the two dimensional array values that has N rows and M columns:

```
for (row=0; row<=N-1; row++) {
      for (col=0; col<=M-1; col++) {
            printf("%d ",
                  values[row][col]);
      }
}
```

Initialization

- Initialize to constant values:

```
int a[4][3] = { {1, 2, 3},
                {4, 5, 6},
                {7, 9, 10},
                {11, 12, 13} };
```

Short Initializers - Not Needed For Class!!!

- Short initializers:
  - Fill first few rows as specified, remaining with 0's:

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

  - Fill first elements of every row as specified, remaining with 0's :

```
int a[4][3] = { {1},
                {4, 5, 6},
                {7, 9},
                {11, 12, 13} };
```

Omitting Braces - Not Needed For Class!!!

- Can omit inner braces:

```
int a[4][3] = {1, 2, 3, 4,
               5, 6, 7, 9,
              10, 11, 12,
              13};
```

- Risky practice: one missed or extra value affects the rest of the initializer

Exercise

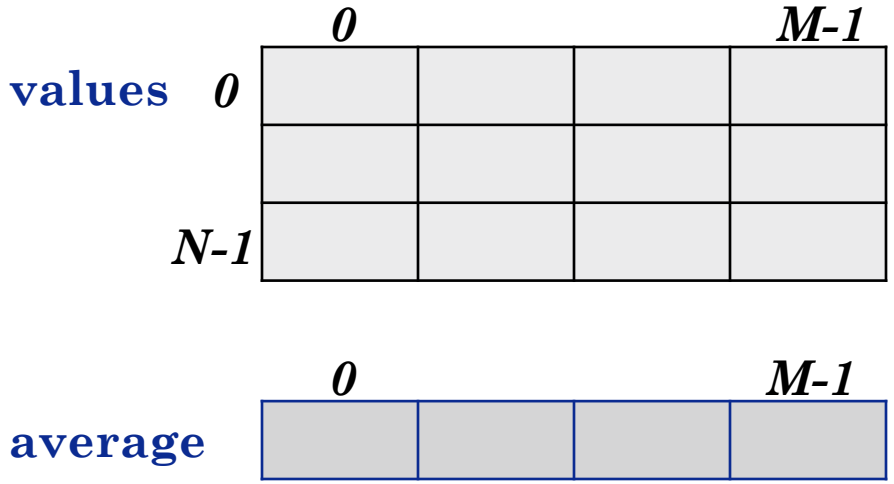
- Write a program that computes the average of every column of a two dimensional array of N rows and M columns:

Solution

```
//Declare variables
int values[N][M];
float average[M];
int sum;

//Read values
      …

//Print content
for (col=0; col<M; col++) {
      sum = 0;
      for (row=0; row<N; row++) {
            sum += values[row][col];
      }
      average[col] = (float) sum/N;
}
```



Variable Length Array (VLA) - Not Needed For Class

- It is also possible to use an expression that is not a constant for the array size

```
int size;

printf("How many elements?\n");

scanf("%d", &size);

int a[size];
```

- VLAs can be multidimensional

```
int a[rows][cols];
```

- Advantage: use correct size instead of guessing, avoid problems with array too short or too long
- Disadvantage (we'll come back to this later in the course)

Copy an Array

- Direct assignment is not applicable

```
int a[5] = {10, 20, 30, 40, 50};
int b[5];
b = a;  //Generates a compilation error!
```

- Use a loop, copy elements one by one (Preferred Method)

```
for (idx=0;idx<5;idx++) {
        b[idx] = a[idx];
}
```

- Use memcpy (memory copy) function in <string.h> (Not Needed For Class, Yet):

```
include <string.h>
        …
memcpy(a, b, sizeof(a));    //faster than a loop
```

Summary

- Array types
  - One dimensional
  - Multi dimensional
- Initialization
- Access and Bounds
- Copying an array