CSc 352

C Programming
2D Arrays

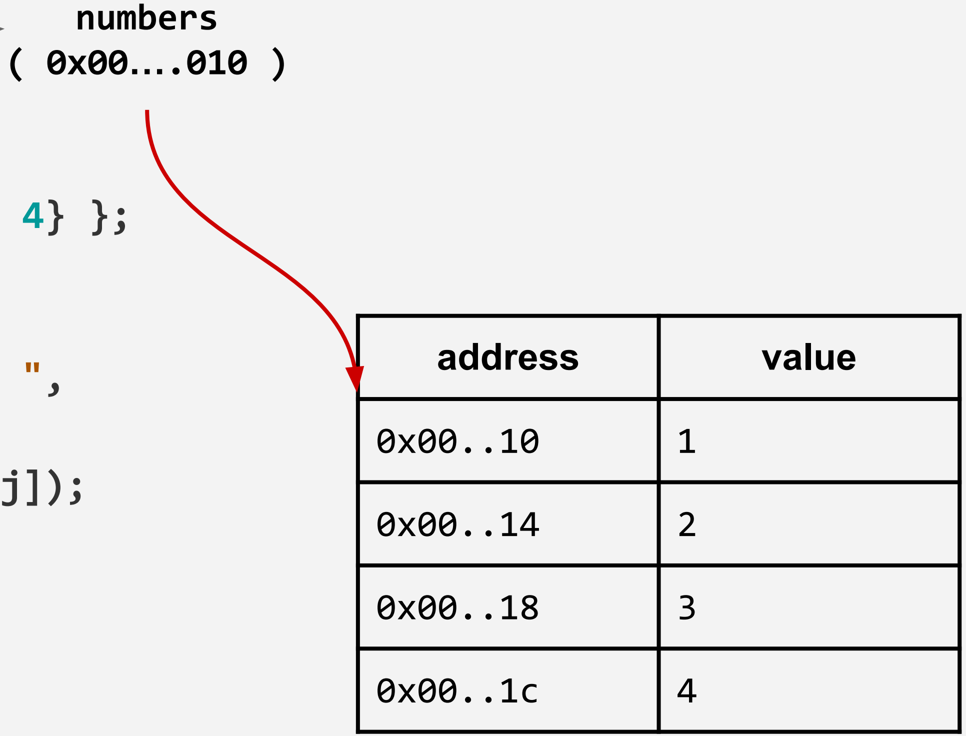
Benjamin Dicken

2D Array

```c
int main() {
  int numbers[2][2] = { {1, 2}, {3, 4} };
  for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
      printf("Element at address %p ",
          &numbers[i][j]);
      printf("is %d\n", numbers[i][j]);
    }
  }
}
```

2D Array

```c
int main() {
 int numbers[2][2] = { {1, 2}, {3, 4} };
 for (int i = 0; i < 2; i++) {
  for (int j = 0; j < 2; j++) {
   printf("Element at address %p ",
     &numbers[i][j]);
   printf("is %d\n", numbers[i][j]);
  }
 }
}
```



numbers
(0x00....010)

address	value
0x00..10	1
0x00..14	2
0x00..18	3
0x00..1c	4

2D Array Memory Layout

The memory layout for a 2D array is one, contiguous block of memory sized **N * M * T** where **N** is the number of rows, **M** is the number of columns, and **T** is the size of the type of each element.

With these “basic” 2D arrays, each row is of same length, even if not given a value explicitly

2D Array Indexing

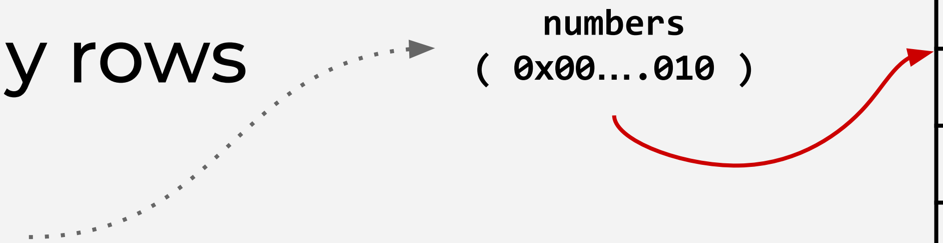
To access an element at a pair of 2D indexes, such as:

```
int array[t][r] = {....}
....
printf("%d", array[x][y]);
```

The program can take the base address of **`array`** and add **`((r*x)+y)`** to get to the address of the requested element.

2D Array rows

```
int main() {
  int numbers[3][4] = { {1, 2, 3, 4},
                {50, 75}, {10, 20, 100} };
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
      printf("Element at address %p ",
        &numbers[i][j]);
      printf("is %d\n", numbers[i][j]);
    }
  }
}
```

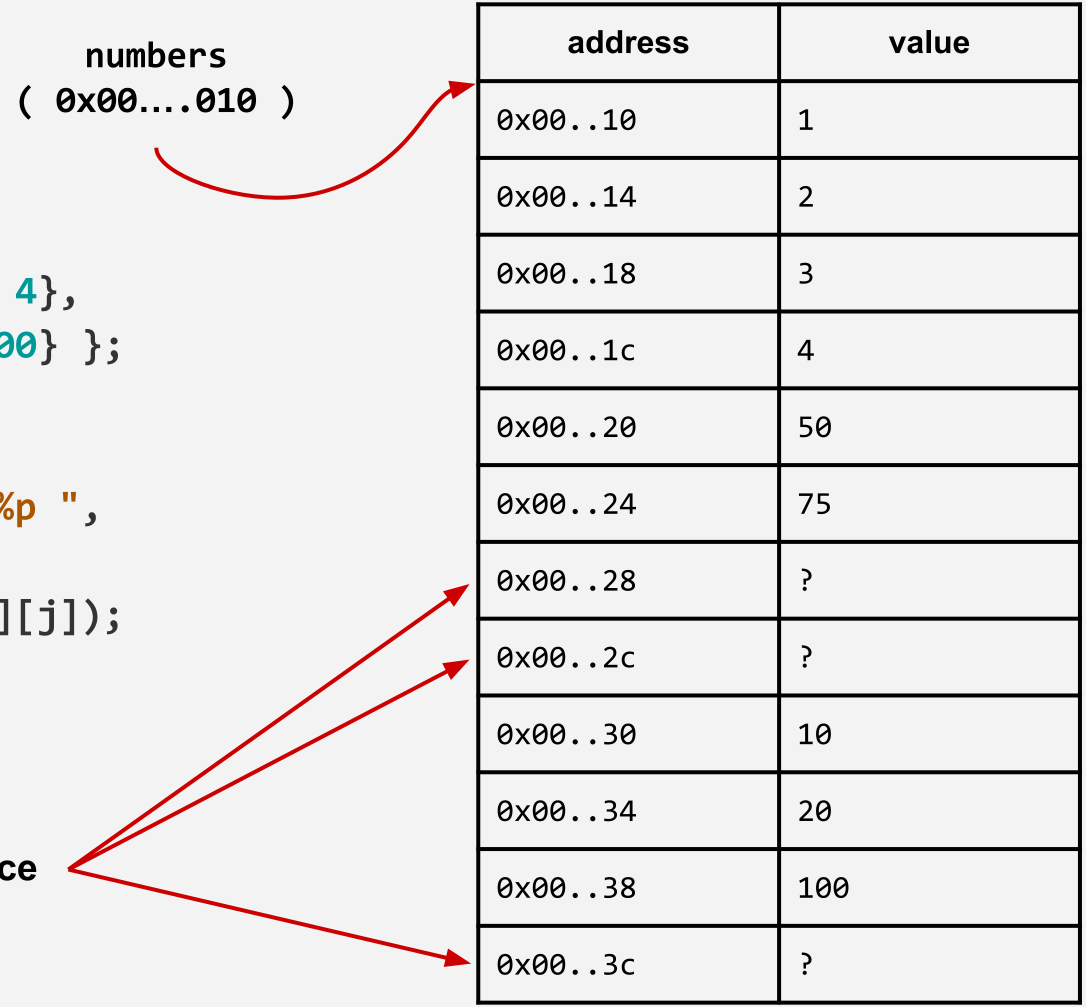


numbers
(0x00....010)

address	value
0x00..10	1
0x00..14	2
0x00..18	3
0x00..1c	4
0x00..20	50
0x00..24	75
0x00..28	?
0x00..2c	?
0x00..30	10
0x00..34	20
0x00..38	100
0x00..3c	?

2D Array rows

```
int main() {
  int numbers[3][4] = { {1, 2, 3, 4},
              {50, 75}, {10, 20, 100} };
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
      printf("Element at address %p ",
        &numbers[i][j]);
      printf("is %d\n", numbers[i][j]);
    }
  }
}
```



numbers
(0x00....010)

address	value
0x00..10	1
0x00..14	2
0x00..18	3
0x00..1c	4
0x00..20	50
0x00..24	75
0x00..28	?
0x00..2c	?
0x00..30	10
0x00..34	20
0x00..38	100
0x00..3c	?

Wasted space

Activity

```
int main() {
  int numbers[3][4] = { {1, 2, 3, 4},
             {50, 75}, {10, 20, 100} };
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
      printf("Element at address %p ",
        &numbers[i][j]);
      printf("is %d\n", numbers[i][j]);
    }
  }
}
```

Say we want some kind of alternative 2D structure to store rows of numbers, but we don't want any wasted space.

How can we fix the "wasted space" issue?

address	value
0x00..10	1
0x00..14	2
0x00..18	3
0x00..1c	4
0x00..20	50
0x00..24	75
0x00..28	?
0x00..2c	?
0x00..30	10
0x00..34	20
0x00..38	100
0x00..3c	?

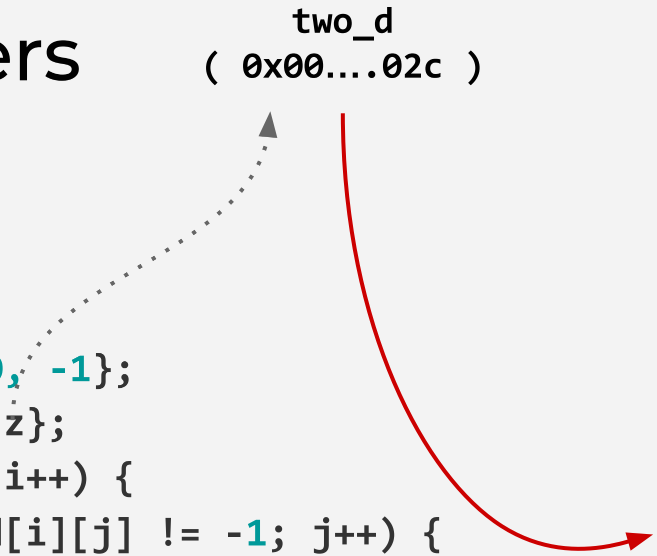
Activity

What will this print?

```
int main() {
  int x[2] = {1, -1};
  int y[1] = {-1};
  int z[4] = {10, 20, 100, -1};
  int* two_d[3] = {x, y, z};
  for (int i = 0; i < 3; i++) {
    for (int j = 0; two_d[i][j] != -1; j++) {
      printf("Element at address %p ",
          &two_d[i][j]);
      printf("is %d\n", two_d[i][j]);
    }
  }
  printf("%p %p %p %p", x, y, z, two_d);
}
```

Assuming that these arrays are placed in sequence on the stack.

Array of Pointers



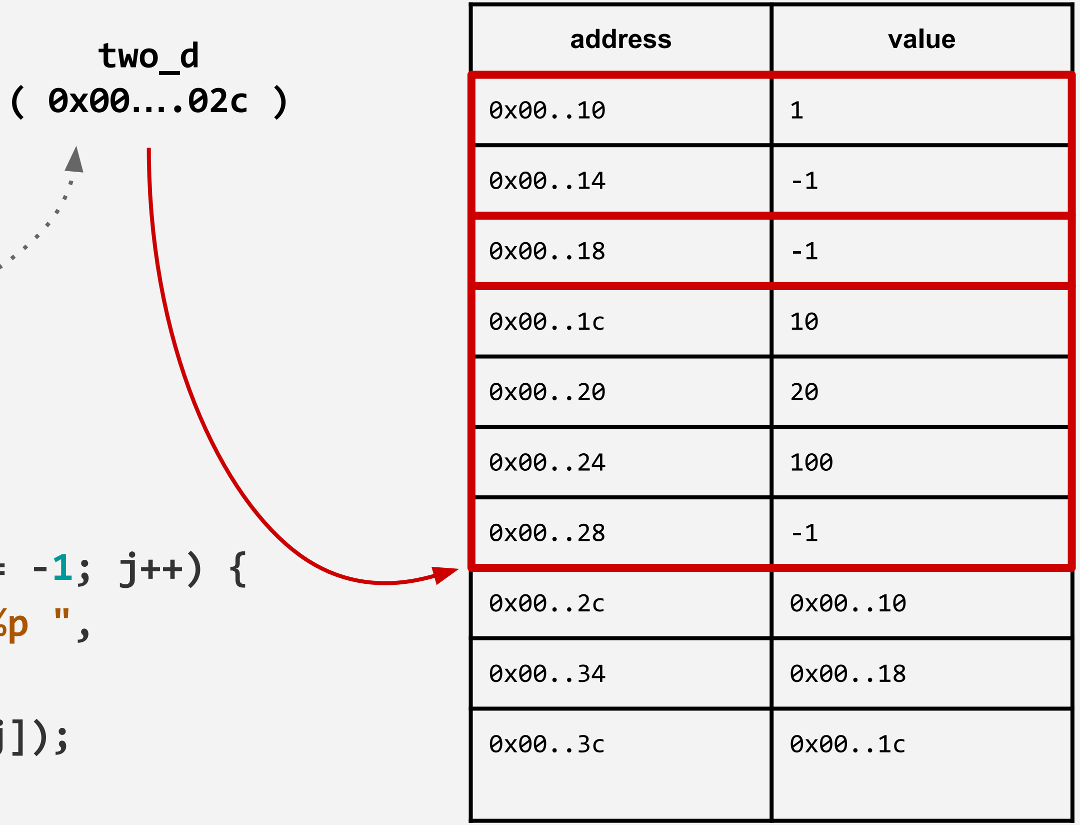
two_d
(0x00....02c)

```c
int main() {
  int x[2] = {1, -1};
  int y[1] = {-1};
  int z[4] = {10, 20, 100, -1};
  int* two_d[3] = {x, y, z};
  for (int i = 0; i < 3; i++) {
    for (int j = 0; two_d[i][j] != -1; j++) {
      printf("Element at address %p ",
          &two_d[i][j]);
      printf("is %d\n", two_d[i][j]);
    }
  }
  printf("%p %p %p %p", x, y, z, two_d);
}
```

address	value
0x00..10	1
0x00..14	-1
0x00..18	-1
0x00..1c	10
0x00..20	20
0x00..24	100
0x00..28	-1
0x00..2c	0x00..10
0x00..34	0x00..18
0x00..3c	0x00..1c

Array of Pointers

```c
int main() {
  int x[2] = {1, -1};
  int y[1] = {-1};
  int z[4] = {10, 20, 100, -1};
  int* two_d[3] = {x, y, z};
  for (int i = 0; i < 3; i++) {
    for (int j = 0; two_d[i][j] != -1; j++) {
      printf("Element at address %p ",
          &two_d[i][j]);
      printf("is %d\n", two_d[i][j]);
    }
  }
  printf("%p %p %p %p", x, y, z, two_d);
}
```

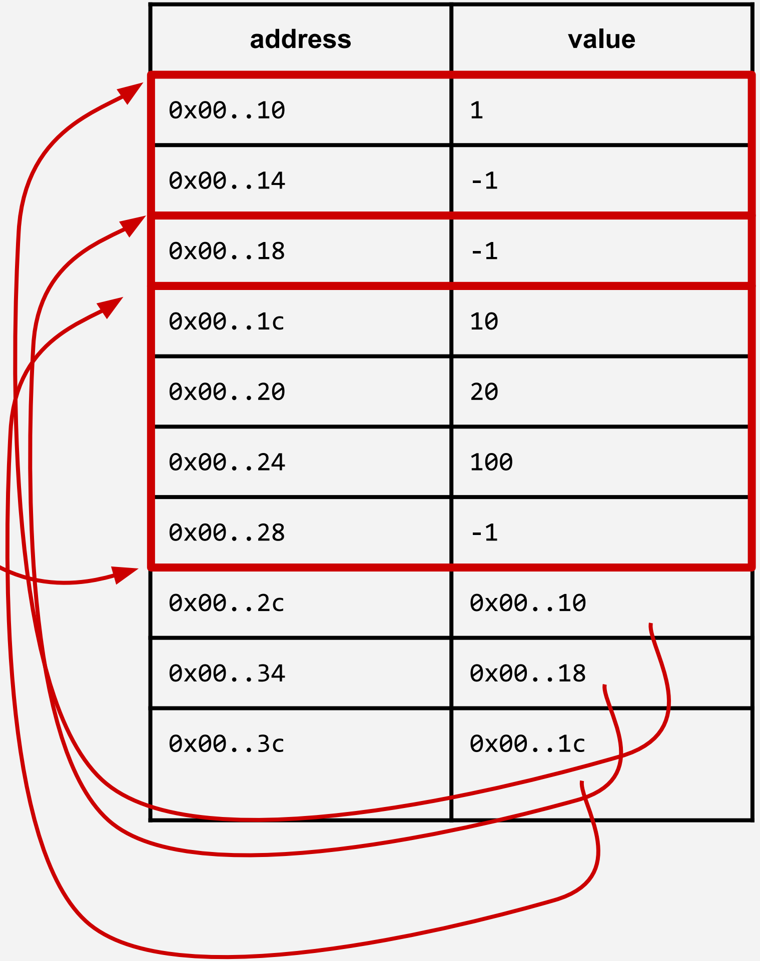


address	value
0x00..10	1
0x00..14	-1
0x00..18	-1
0x00..1c	10
0x00..20	20
0x00..24	100
0x00..28	-1
0x00..2c	0x00..10
0x00..34	0x00..18
0x00..3c	0x00..1c

Array of Pointers

```
int main() {
  int x[2] = {1, -1};
  int y[1] = {-1};
  int z[4] = {10, 20, 100, -1};
  int* two_d[3] = {x, y, z};
  for (int i = 0; i < 3; i++) {
    for (int j = 0; two_d[i][j] != -1; j++) {
      printf("Element at address %p ",
          &two_d[i][j]);
      printf("is %d\n", two_d[i][j]);
    }
  }
  printf("%p %p %p %p", x, y, z, two_d);
}
```

two_d
(0x00....02c)



address	value
0x00..10	1
0x00..14	-1
0x00..18	-1
0x00..1c	10
0x00..20	20
0x00..24	100
0x00..28	-1
0x00..2c	0x00..10
0x00..34	0x00..18
0x00..3c	0x00..1c

```
int numbers[3][4] =
  { {1, 2}, {50}, {10, 20, 100, 1} };
```

address	value
0x00..10	1
0x00..14	2
0x00..18	0
0x00..1c	0
0x00..20	50
0x00..24	0
0x00..28	0
0x00..2c	0
0x00..30	10
0x00..34	20
0x00..38	100
0x00..3c	1

```
int x[2] = {1, -1};
int y[1] = {-1};
int z[4] = {10, 20, 100, -1};
int* two_d[3] = {x, y, z};
```

address	value
0x00..10	1
0x00..14	-1
0x00..18	-1
0x00..1c	10
0x00..20	20
0x00..24	100
0x00..28	-1
0x00..2c	0x00..10
0x00..34	0x00..18
0x00..3c	0x00..1c

Activity

Analyze the program

Find **/tmp/story.txt** and **/tmp/most_occurring.c**

Reads in paragraphs, determined most occurring word for each, uses 2D arrays

1. Copy it to a directory that you control
   ```
   $ cp /tmp/story.txt ~/
   $ cp /tmp/most_occurring.c ~/
   ```
2. Compile and try it out!
   ```
   $ gcc .... most_occurring.c
   $ cat story.txt | ./a.out
   ```
3. Look at the source, what are the weaknesses?

story.txt

```
There once was a bear
that lived by the sea in a tiny house.

He wanted to go into town to get some
ice-cream. However, he knew people in town
would be scared of him. Those town people
are scared easily.

He came up with a plan to get around
this. The plan was to dress up as a man.
The plan worked, and he was able to get
ice-cream.
```

output

```
Most-occurring word from paragraph 1: a
Most-occurring word from paragraph 2: town
Most-occurring word from paragraph 3: plan
```