CSE 1320

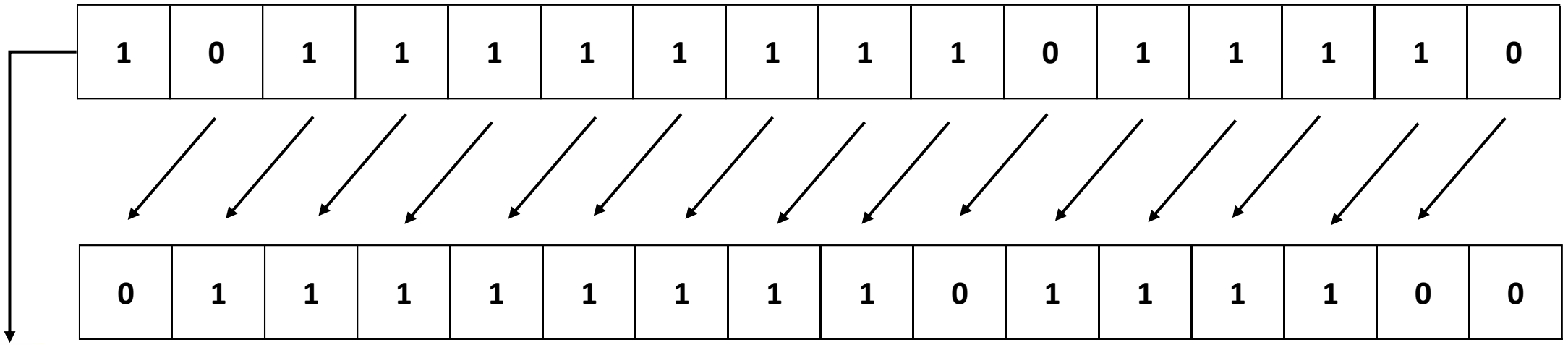
Week of 02/13/2023

Instructor : Donna French

Bit Shifting

left shift

```
expression1 << expression 2
```

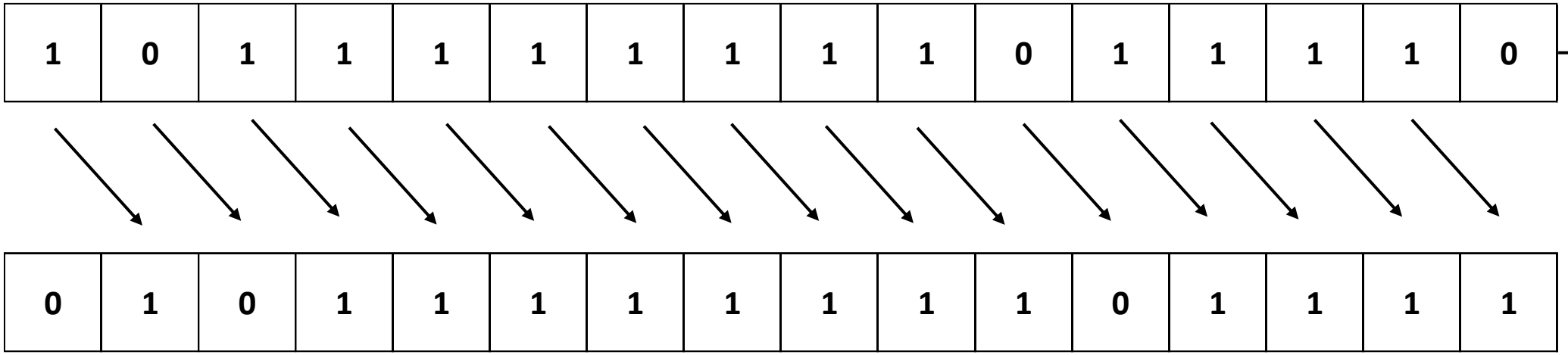


Bitbucket

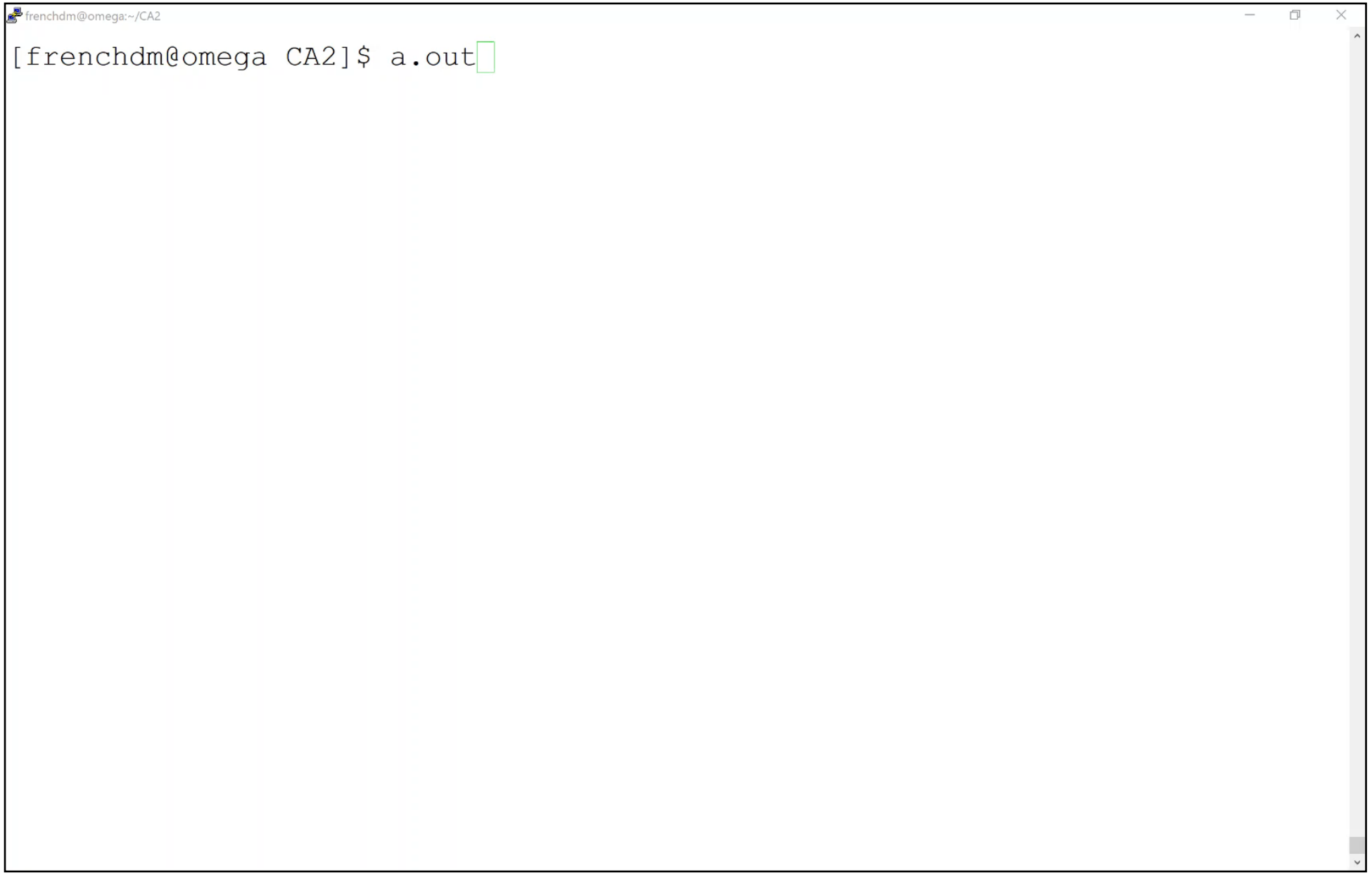
Bit Shifting

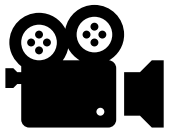
right shift

```
expression1 >> expression 2
```



Bitbucket


frenchdm@omega:~/CA2
[frenchdm@omega CA2]$ a.out



Using >> for Division

>> 0	2^0	Divide by 1	100 >> 0 = 100
>> 1	2^1	Divide by 2	100 >> 1 = 50
>> 2	2^2	Divide by 4	100 >> 2 = 25
>> 3	2^3	Divide by 8	100 >> 3 = 12
>> 4	2^4	Divide by 16	100 >> 4 = 6
>> 5	2^5	Divide by 32	100 >> 5 = 3
>> 6	2^6	Divide by 64	100 >> 6 = 1
>> 7	2^7	Divide by 128	100 >> 7 = 0

Using << for Multiplication

<< 0	2^0	Multiple by 1	1 << 0 = 1
<< 1	2^1	Multiple by 2	1 << 1 = 2
<< 2	2^2	Multiple by 4	1 << 2 = 4
<< 3	2^3	Multiple by 8	1 << 3 = 8
<< 4	2^4	Multiple by 16	1 << 4 = 16
<< 5	2^5	Multiple by 32	1 << 5 = 32
<< 6	2^6	Multiple by 64	1 << 6 = 64
<< 7	2^7	Multiple by 128	1 << 7 = 128

Random Number Generator

```
int rand(void)
```

returns a pseudo-random number in the range of 0 to `RAND_MAX`

`RAND_MAX` is a constant whose default value may vary between implementations but it is granted to be at least 32767.

Random Number Generator

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i;



    for (i = 0; i < 10; i++)
    {
        printf("%d\n", rand() % 50);
    }

    return 0;
}
```

```
[frenchdm@omega ~]$ a.out
33
36
27
15
43
35
36
42
49
21
```

Random Number Generator

To generate random numbers that change between calls to the `rand()` function, call `srand()` to seed the random number generator

Pass `time(0)` to `srand()` to seed the random number generator with the number of seconds since the Epoch (00:00:00 UTC, January 1, 1970).

This will generate a different set of random numbers between calls.

Random Number Generator

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i;

    srand(time(0));

    for (i = 0; i < 10; i++)
    {
        printf("%d\n", rand() % 50);
    }

    return 0;
}
```

```
[frenchdm@omega ~]$ a.out
49
22
35
34
48
27
32
13
14
39
```

Random Number Generator

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
   int i = 0;

   srand(time(0));

   for (i = 0; i < 10; i++)
   {
      printf("%d\n", rand() % 50);
   }

   return 0;
}
```

Must include `stdlib.h` to use `rand()` and `time.h` to use `time(0)`

Produces a random number between 0 and 49 (includes 0 and 49)

```
./a.out
20
4
47
4528
31
43
0
20
25
./a.out
15
49
37
10
42
49
41
12
33
7
```

Random Number Generator

How to find a random number within a range?

For example, find a random number inclusively between 32 and 87.

```
printf("%d\n", rand() % (end-start));
```

This will result in a random number between 0 and 54

87-32 is 55 but when we mod(%) by 55, we get everything less than 55

Random Number Generator

Our lower bound is 32 so let's add 32 to our random number.

```
printf("%d\n", rand() % (end-start) + start);
```

Now we have (0+32) and (54+32) which is 32 to 86.

We need 32 to 87.

So let's expand our mod value by 1

Random Number Generator

So let's expand our mod value by 1

```
printf("%d\n", rand() % (end-start+1) + start);
```

So if `start` is 32 and `end` is 87

```
rand() % (end-start+1) + start
rand() % (87-32+1) + 32
rand() % (56) + 32
```

`rand() % 56` will give us number from 0 to 55 so when we add `start`(32), we will get numbers from 32 to 87.

Random Number Generator

Let's check our formula with a few more examples…

```
printf("%d\n", rand() % (end-start+1) + start);
```

Random number between 1 and 15

`rand() % (15-1+1)` gives 0 to 14 which will be 1 to 15 when `start`(1) is added

Random number between 23 and 61

`rand() % (61-23+1)` gives 0 to 38 which will be 23 to 61 when `start`(23) is added

Random Number Generator

Remember that randomly generated numbers are **NOT** unique.

27	20	1
23	17	35
23	31	19
15	33	14
42	17	24
34	6	43
25	17	31
8	46	21
23	38	17
13	37	1

One Dimensional Arrays

Arrays

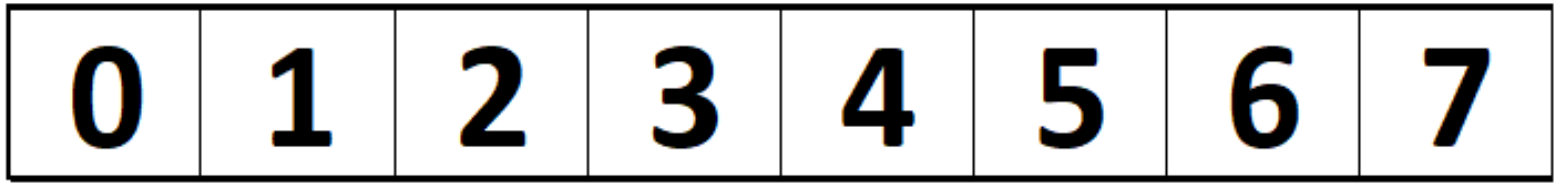
- aggregate type
- used to store collections of related data
- multiple values of the same data type can be stored with one variable name

One Dimensional Arrays

each data object occupies one cell of the array

spice rack

index




```
int spice_rack[8]
```

- each cell is type `int`
- 8 cells
- array name is `spice_rack`

- indicies are 0 through 7
  - must be integers
  - first cell is always 0
  - last cell is 1 less than the total number of cells

Initialization of Arrays

Arrays can be initialized 2 ways

Method 1 - the array declaration

- comma separated list enclosed in braces
- initial values can be constants or expression using declared and initialized variables

```
int MyArray[10] = {12,42,63,48,59,62,77,82,91,10};
char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};
char MyCharArray[] = {"ABCDEFGHIJ"};
```

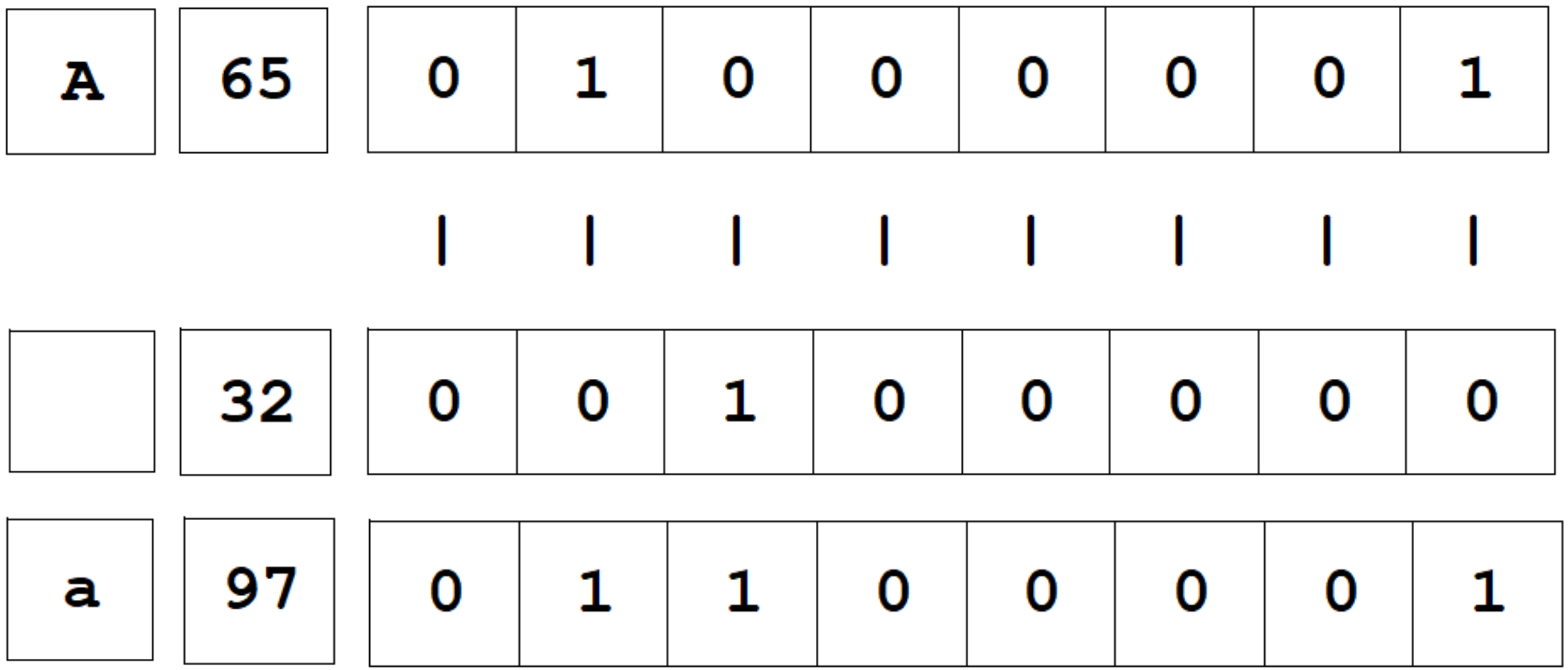
```
int i;
int MyArray[10] = {12,42,63,48,59,62,77,82,91,10};
int Choice;
char MyCharArray[10];

for (i = 0; i < 10; i++)
{
  printf("MyArray[%d] = %d\n", i, MyArray[i]);
}

printf("Which array element do you want to see? ");
scanf("%d", &Choice);
printf("The value of array element %d is %d\n", Choice, MyArray[Choice]);

for (i = 0; i < 10; i++)
{
  MyArray[i] = MyArray[i] >> 1;
  printf("MyArray[%d] = %d\n", i, MyArray[i]);
}

printf("Which array element do you want to see? ");
scanf("%d", &Choice);
printf("The value of array element %d is %d\n", Choice, MyArray[Choice]);
```

```
MyArray[0] = 12
MyArray[1] = 42
MyArray[2] = 63
MyArray[3] = 48
MyArray[4] = 59
MyArray[5] = 62
MyArray[6] = 77
MyArray[7] = 82
MyArray[8] = 91
MyArray[9] = 10
Which array element do you want to see? 7
The value of array element 7 is 82
MyArray[0] = 6
MyArray[1] = 21
MyArray[2] = 31
MyArray[3] = 24
MyArray[4] = 29
MyArray[5] = 31
MyArray[6] = 38
MyArray[7] = 41
MyArray[8] = 45
MyArray[9] = 5
Which array element do you want to see? 0
The value of array element 0 is 6
```

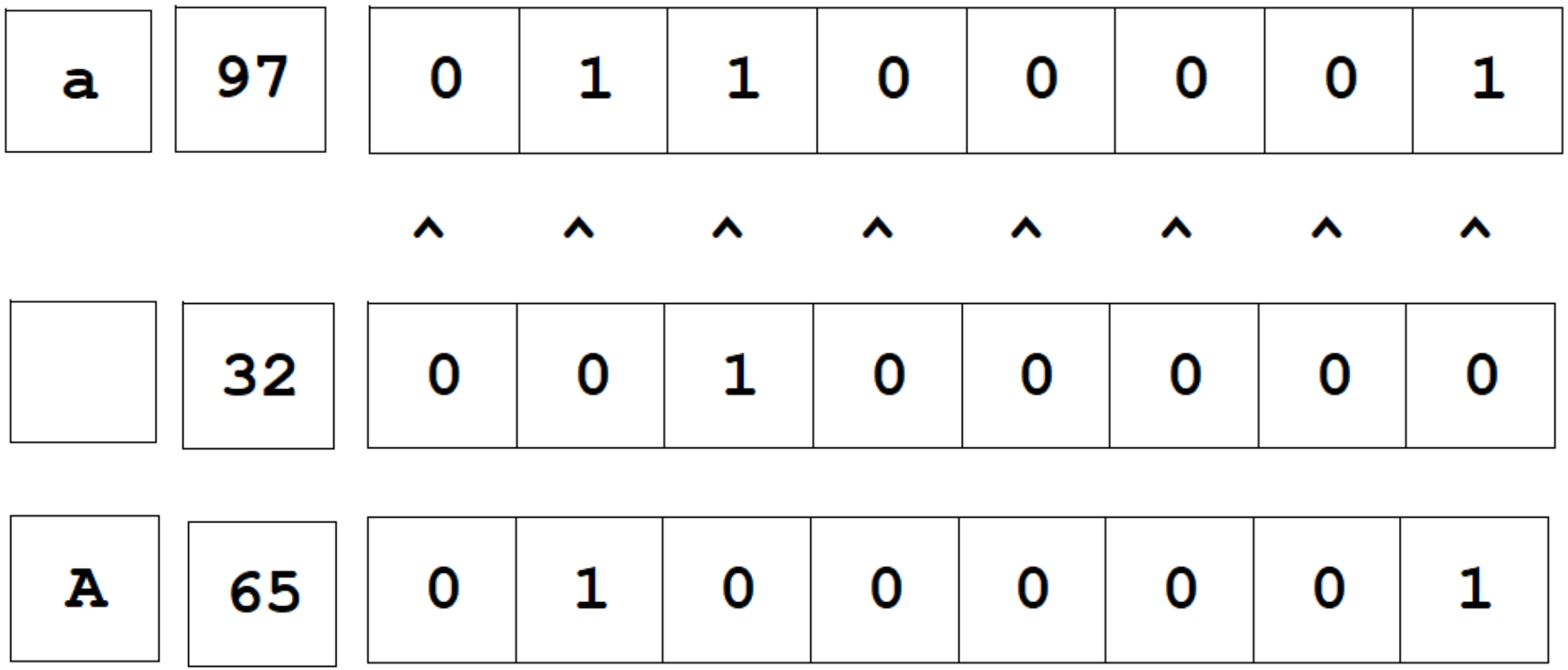
```
int main(void)
{
	int i;
	int Choice;
	char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};

	for (i = 0; i < 10; i++)
	{
		printf("MyCharArray[%d] = %c\n", i, MyCharArray[i]);
	}

	printf("Which array element do you want to see? ");
	scanf("%d", &Choice);
	printf("The value of array element %d is %c\n",
		 Choice, MyCharArray[Choice]);

	for (i = 0; i < 10; i++)
	{
		MyCharArray[i] = MyCharArray[i] | 32;
		printf("MyCharArray[%d] = %c\n", i, MyCharArray[i]);
	}

	return 0;
}
```

```
MyCharArray[0] = A
MyCharArray[1] = B
MyCharArray[2] = C
MyCharArray[3] = D
MyCharArray[4] = E
MyCharArray[5] = F
MyCharArray[6] = G
MyCharArray[7] = H
MyCharArray[8] = I
MyCharArray[9] = J
Which array element do you want to see? 5
The value of array element 5 is F
MyCharArray[0] = a
MyCharArray[1] = b
MyCharArray[2] = c
MyCharArray[3] = d
MyCharArray[4] = e
MyCharArray[5] = f
MyCharArray[6] = g
MyCharArray[7] = h
MyCharArray[8] = i
MyCharArray[9] = j
```

```
char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};

MyCharArray[i] = MyCharArray[i] | 32;
```



```
char MyCharArray[10] = {'a','b','c','d','e','f','g','h','i','j'};

MyCharArray[i] = MyCharArray[i] ^ 32;
```



Array Initialization

```
Breakpoint 1, main () at array3Demo.c:9
9           char MyCharArray[10] = {'A','B','C','D','E','F','G','H','I','J'};
(gdb) p MyCharArray
$1 = "\340\005@\000\000\000\000\000\000"
(gdb) step
11              for (i = 0; i < 10; i++)
(gdb) p MyCharArray
$2 = "ABCDEFGHIJ"
```

```
Breakpoint 1, main () at array4Demo.c:9
9           char MyCharArray[] = {"ABCDEFGHIJ"};
(gdb) p MyCharArray
$3 = "\340\005@\000\000\000\000\000\000\000"
(gdb) step
11              for (i = 0; i < 10; i++)
(gdb) p MyCharArray
$4 = "ABCDEFGHIJ"
```

array3Demo.c
array4Demo.c

Array Initialization

Method 2 - in the executable code

```
for (i = 0; i < 10; i++)
{
    printf("Enter value for MyArray[%d] ", i);
    scanf("%d", &MyArray[i]);
}
```

```c
int main(void)
{
  int i;
  int MyArray[10];
  int Choice;

  for (i = 0; i < 10; i++)
  {
    printf("Enter value for MyArray[%d] ", i);
    scanf("%d", &MyArray[i]);
  }

  for (i = 0; i < 10; i++)
  {
    printf("MyArray[%d] = %d\n", i, MyArray[i]);
  }

  printf("\nEnter array element to display? ");
  scanf("%d", &Choice);
  printf("\nArray element %d is %d\n",
         Choice, MyArray[Choice]);

  return 0;
}
```

```
Enter value for MyArray[0] 4
Enter value for MyArray[1] 5
Enter value for MyArray[2] 22
Enter value for MyArray[3] 77
Enter value for MyArray[4] 11
Enter value for MyArray[5] 33
Enter value for MyArray[6] 98
Enter value for MyArray[7] 3
Enter value for MyArray[8] 56
Enter value for MyArray[9] 23
MyArray[0] = 4
MyArray[1] = 5
MyArray[2] = 22
MyArray[3] = 77
MyArray[4] = 11
MyArray[5] = 33
MyArray[6] = 98
MyArray[7] = 3
MyArray[8] = 56
MyArray[9] = 23

Enter array element to display? 6

Array element 6 is 98
```

array1Demo.c

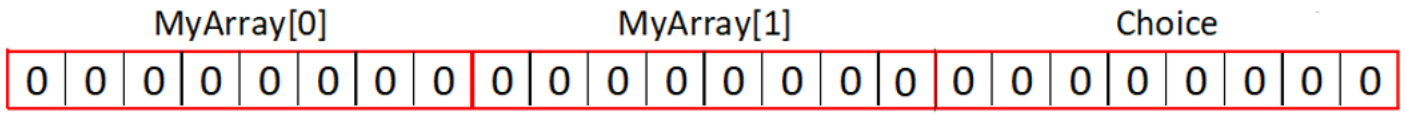
Array Initialization

```
int i;
int Choice = 0;
int MyIntArray[2] = {0,0};

printf("Choice is currently %d at %p\t", Choice, &Choice);

for (i = 0; i <= 2; i++)
{
   MyIntArray[i] = i;
   printf("MyIntArray[%d] = %d\t%p\n", i, MyIntArray[i], &MyIntArray[i]);
   getchar();
   printf("Choice is currently %d at %p\t", Choice, &Choice);
}
```

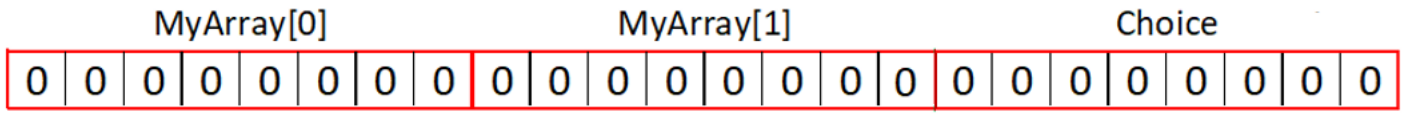
Array Initialization

```
Choice is currently 0 at 0x7fff58751b98 MyIntArray[0] = 0       0x7fff58751b90

Choice is currently 0 at 0x7fff58751b98 MyIntArray[1] = 1       0x7fff58751b94

Choice is currently 0 at 0x7fff58751b98 MyIntArray[2] = 2       0x7fff58751b98

Choice is currently 2 at 0x7fff58751b98
```

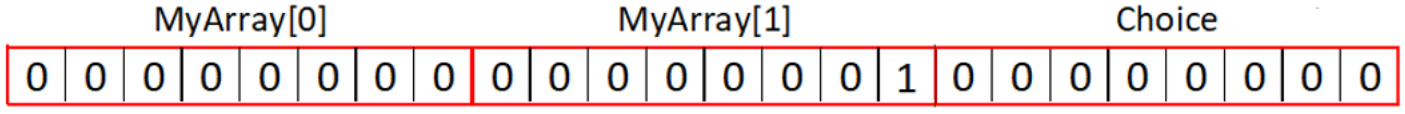
Initialization of Arrays – Out of Bounds



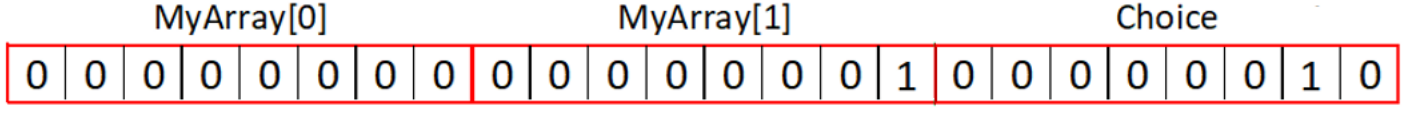
Assign 0 to MyArray[0]



Assign 1 to MyArray[1]



Assign 2 to MyArray[2]



Storing and Printing with Arrays

```
#include <stdio.h>

int main(void)
{
	char MyClassName[10] = {};

	printf("What is your class's name ");
	scanf("%s", MyClassName);

	printf("%s\n", MyClassName);

	return 0;
}
```

```
[frenchdm@omega ~]$ a.out
What is your class's name CSE1320
CSE1320
[frenchdm@omega ~]$
```

Arrays as Parameters to Functions

Passing an array as a parameter to a function

- calling the function
  - when the name of the array is used without brackets, the array name is evaluated as the address of the array
  - passing the address of the array allows the function to access the elements
  - not possible to pass a copy of the array as a parameter

```
MyFunction(MyIntArray);
```

Arrays as Parameters to Functions

Passing an array as a parameter to a function

- array must be declared in the function header
- array may be accessed in the function code

```
void MyFunction(int MyIntArray[])
{
      printf("Element 0 of MyIntArray is %d", MyIntArray[0]);
}
```

Arrays as Parameters to Functions

Passing an array as a parameter to a function

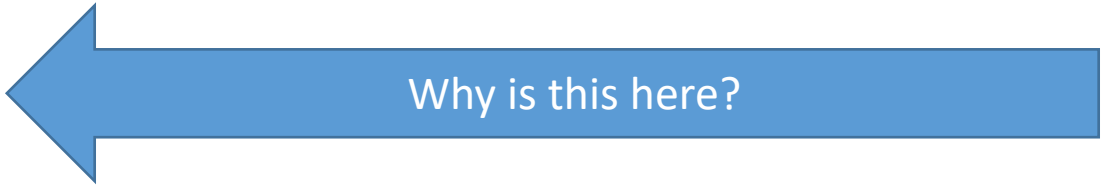
- array must be declared in the function prototype

```
int MyFunction(int MyIntArray[]);
```

Formal parameter name is `MyIntArray` and it is of type `int`.

There is no indication of the number of elements in the parameter. This will not cause an error because the prototype does not cause any memory to be allocated (not a variable definition). Function only knows the address and element type of the array.

```
int i;
int ElementsToEnter;
char MyCharArray[20] = {};

printf("How many characters do you want to enter? ");
scanf("%d", &ElementsToEnter);

getchar();

for (i = 0; i < ElementsToEnter; i++)
{
   printf("Enter character %d ", i);
   MyCharArray[i] = getchar();
   getchar();
}

printf("\n\nThe ASCII sum of the entered "
       "characters is %d\n\n",
       PrintArray(MyCharArray, i));
```



```
int PrintArray(char MyCharArray[], int ElementCount)
{
   int i, ASCIIsum = 0;

   for (i = 0; i < ElementCount; i++)
   {
      printf("MyCharArray[%d] = %c which is ASCII %d\n",
              i, MyCharArray[i], MyCharArray[i]);

      ASCIIsum += MyCharArray[i];
   }

   return ASCIIsum;
}
```

```
printf("\n\nThe ASCII sum of the entered "
       "characters is %d\n\n",
       PrintArray(MyCharArray, i));
```

```
How many characters do you want to enter? 4
Enter character 0 H
Enter character 1 E
Enter character 2 L
Enter character 3 P
MyCharArray[0] = H which is ASCII 72
MyCharArray[1] = E which is ASCII 69
MyCharArray[2] = L which is ASCII 76
MyCharArray[3] = P which is ASCII 80


The ASCII sum of the entered characters is 297
```

Arrays as Parameters to Functions

Can I return an array from a function via the return statement?

Can I create an array inside a function, put data in it and then use

```
return MyArray;
```

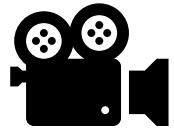
to return the filled up array back to `main()`?

*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\passarray3Demo.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

MyFirstCProgram.c | win801ca.c | ICQ2.c | new 1 | array7Demo.c | passarray2Demo.c | passarray3Demo.c

```
1  // Passing Array3 Demo
2  #include <stdio.h>
3
4
5  int main(void)
6  {
7
8
9      return 0;
10 }
11
12
13
14
15
```



*C:\Users\Donna\Desktop\UTA\Programs\CSE1320\passarray3Demo.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

MyFirstCProgram.c | win801ca.c | ICQ2.c | new 1 | array7Demo.c | passarray2Demo.c

```c
// Passing Array3 Demo
#include <stdio.h>

int CreateMonsterFunctionFunction(void);

int main(void)
{
    int Monster[5];

    Monster = CreateMonsterFunction();

    return 0;
}


int CreateMonsterFunction(void)
{
    int Monster[5] = {0};
    int i = 0;

    for (i = 0; i < 5; i++)
    {
        Monster[i] = i;
    }

    return Monster;
}
```

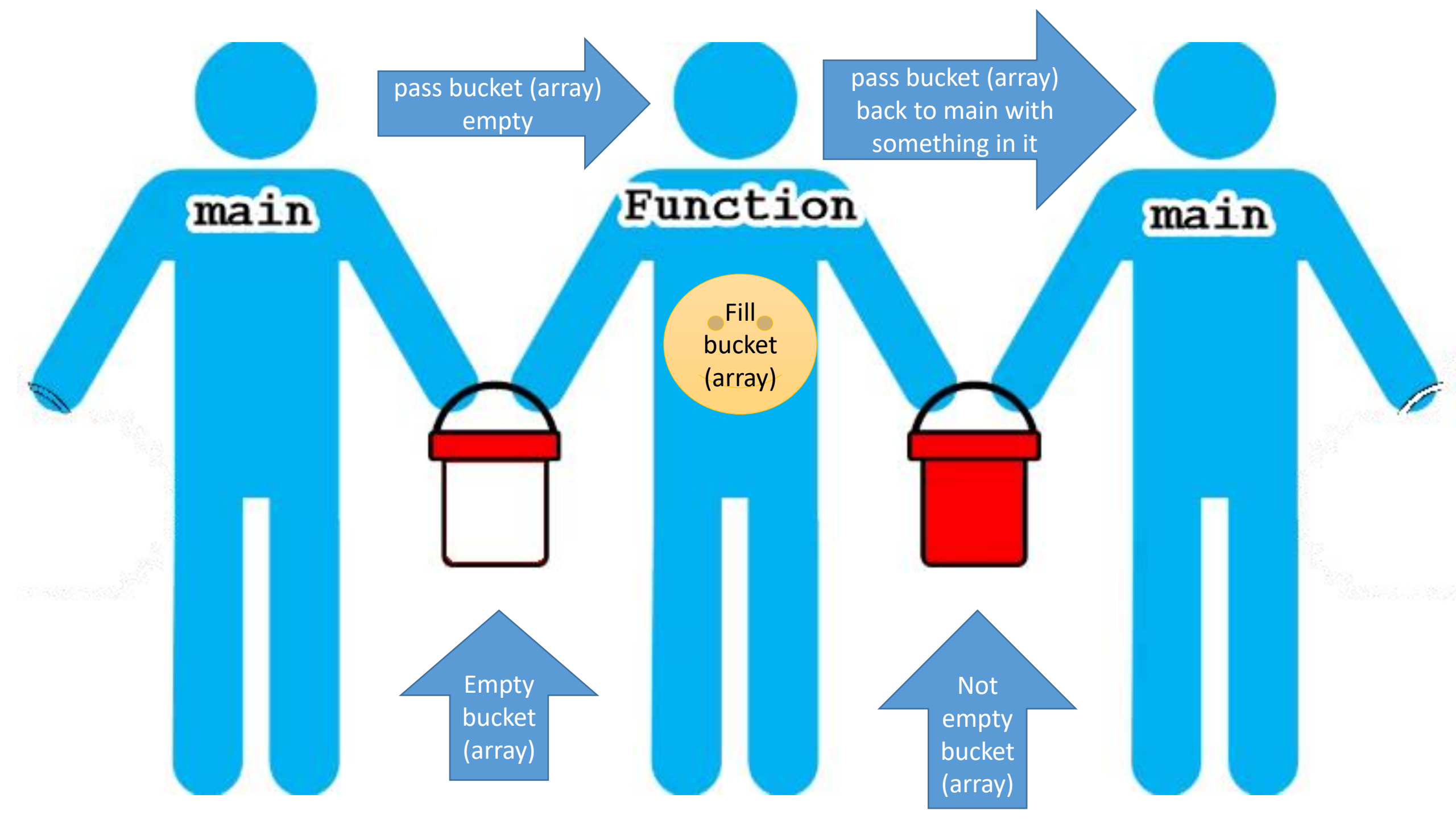
frenchdm@omega:~

```
[frenchdm@omega ~]$ gcc passarray3Demo.c
passarray3Demo.c: In function 'main':
passarray3Demo.c:10: error: incompatible types in assignment
passarray3Demo.c: In function 'CreateMonsterFunction':
passarray3Demo.c:25: warning: return makes integer from pointer without a cast
passarray3Demo.c:25: warning: function returns address of local variable
[frenchdm@omega ~]$
```

```c
// Passing Array3 Demo
#include <stdio.h>

int [] CreateMonsterFunctionFunction(void);
```

```
[frenchdm@omega ~]$ gcc passarray3Demo.c
passarray3Demo.c:4: error: expected identifier or ‘(’ before ‘[’ token
passarray3Demo.c: In function ‘main’:
passarray3Demo.c:10: error: incompatible types in assignment
passarray3Demo.c: At top level:
passarray3Demo.c:16: error: expected identifier or ‘(’ before ‘[’ token
[frenchdm@omega ~]$
```

```c
int [] CreateMonsterFunction(void)
{
    int Monster[5] = {0};
    int i = 0;

    for (i = 0; i < 5; i++)
    {
        Monster[i] = i;
    }

    return Monster;
}
```


pass bucket (array) empty
pass bucket (array) back to main with something in it
main
Function
main
Fill bucket (array)
Empty bucket (array)
Not empty bucket (array)

Passing Arrays

```
int main(void)
{
    int Lion[5];
    char Tiger[5];

    PassArrayFunction(Lion, Tiger);
    PrintArrayFunction(Tiger, Lion);

    return 0;
}
```

passarray2Demo.c

```
PassArrayFunction(Lion, Tiger);
```

```
void PassArrayFunction(int Grizzly[], char Polar[])
{
      int Bear;

      Grizzly[0] = UCHAR_MAX;
      Polar[0] = 'A';

      for (Bear = 1; Bear < 5; Bear++)
      {
            Grizzly[Bear] = Grizzly[Bear-1] >> 1;

            Polar[Bear] = Polar[Bear-1]+1;
      }

      return;
}
```

passarray2Demo.c

```
PrintArrayFunction(Tiger, Lion);
```

```
void PrintArrayFunction(char African[], int Asian[])
{
  int Elephant;

  for (Elephant = 0; Elephant < 5; Elephant++)
  {
    printf("African[%d] = %c\tAsian[%d] = %d\t\t",
           Elephant, African[Elephant],
           Elephant, Asian[Elephant]);

    printf("%d\n", (Asian[Elephant] & 16) ? 1 : 0);
  }
}
```

```
African[0] = A  Asian[0] = 255          1
African[1] = B  Asian[1] = 127          1
African[2] = C  Asian[2] = 63           1
African[3] = D  Asian[3] = 31           1
African[4] = E  Asian[4] = 15           0
```

passarray2Demo.c

Compiler Warning and Overflow

cw1Demo.c

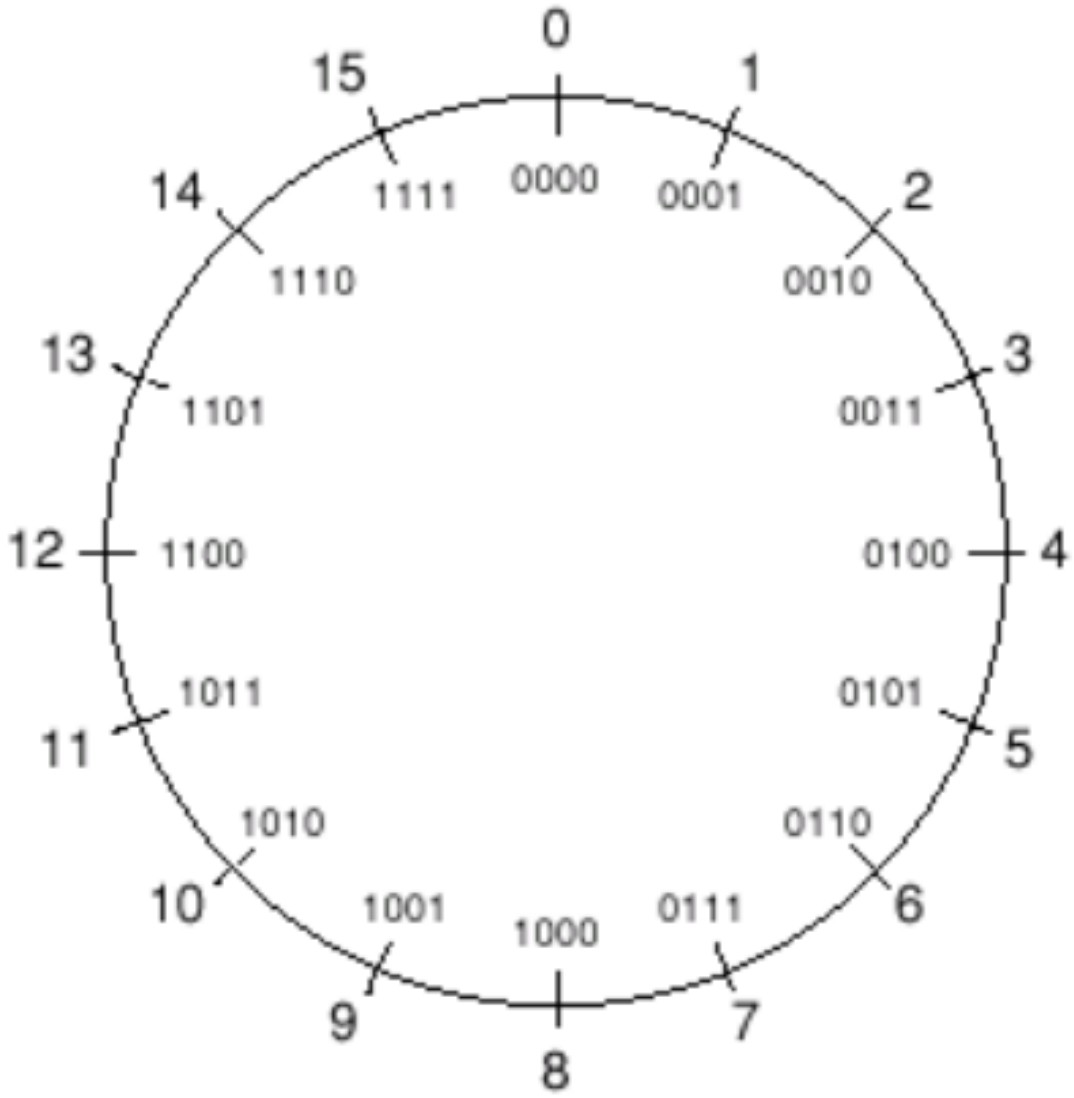
```
#include <stdio.h>
#include <limits.h>

int main(void)
{
  short VarA;

  printf("Enter a value ");
  scanf("%d", &VarA);

  if (VarA > SHRT_MAX)
  {
    printf("VarA is larger than a max short\n");
  }

  printf("\nYou entered %d\n\n", VarA);
}
```

SHRT_MAX is 32767

4-bit Adding in Binary

$$\begin{array}{r} 0110_2 \\ +\ 1001_2 \\ \hline 1111_2 \end{array} \qquad \begin{array}{r} 0101_2 \\ +\ 1101_2 \\ \hline 0010_2 \end{array}$$

$$\begin{array}{r} 0101_2 \\ +\ 1001_2 \\ \hline 1110_2 \end{array} \qquad \begin{array}{r} 1101_2 \\ +\ 1101_2 \\ \hline 1010_2 \end{array}$$



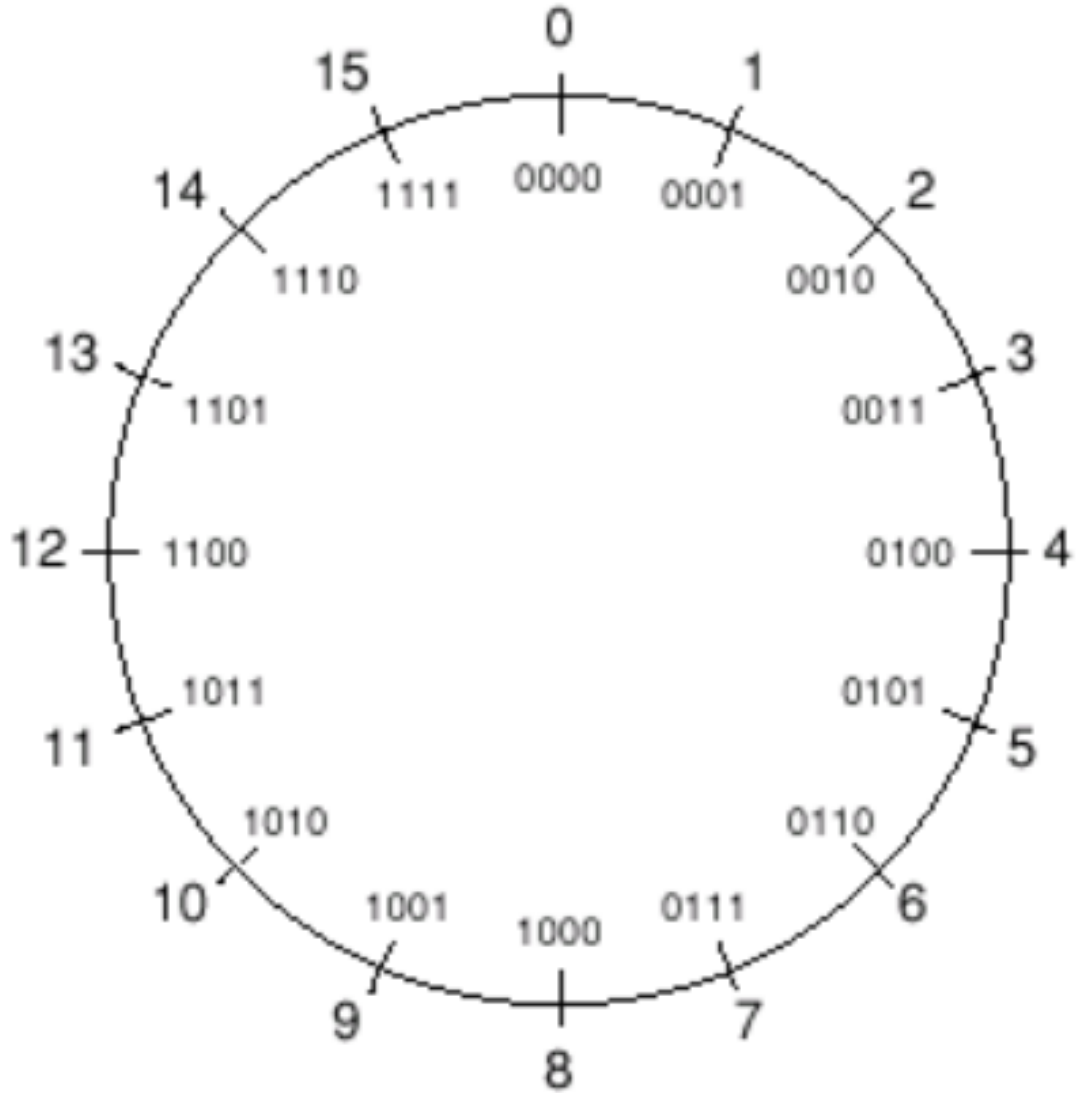
Two's Complement

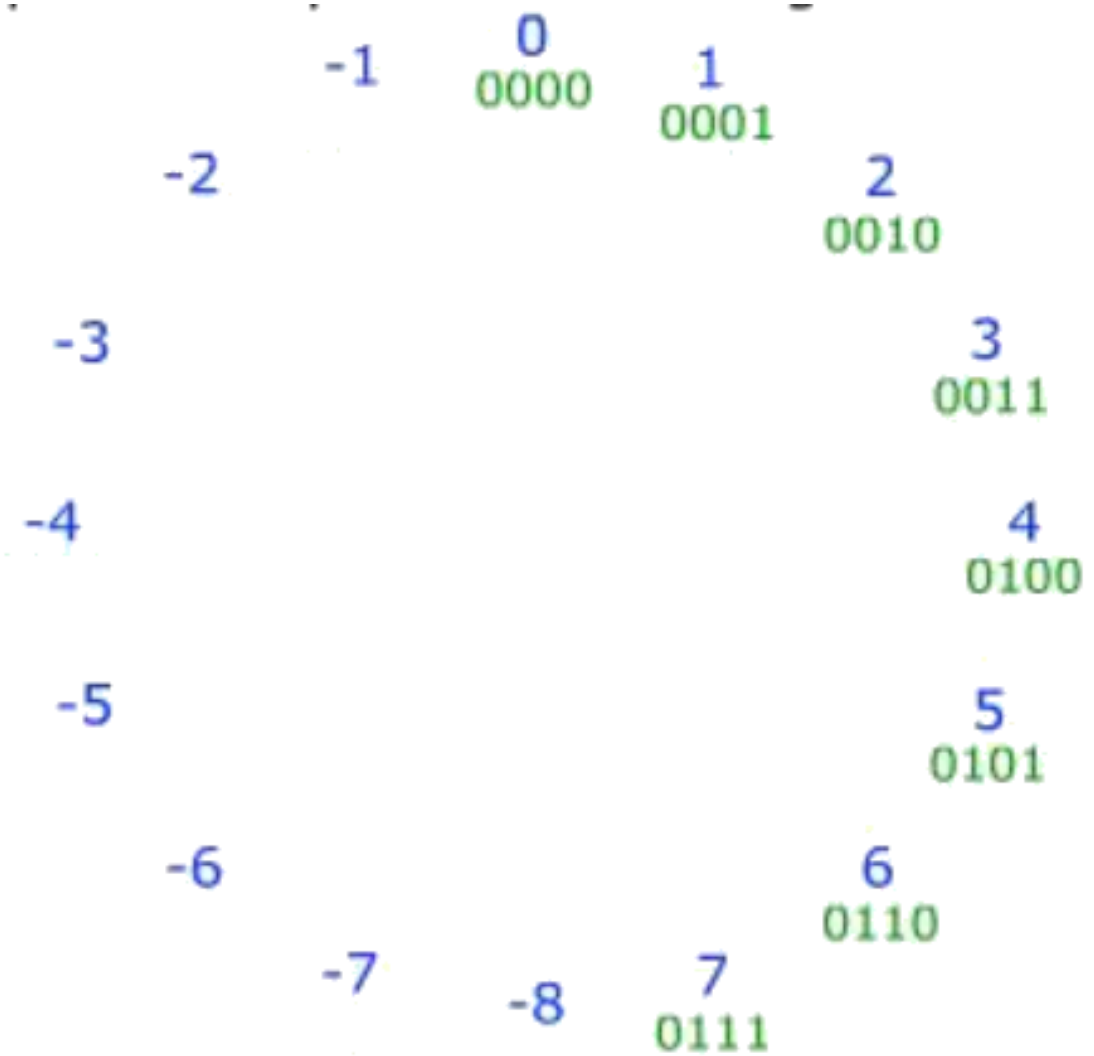
How can we represent negative numbers in binary?

There are several ways

- using a sign bit
- one's complement
- two's complement

Two's complement is the most commonly used technique because it's relatively easy to implement in hardware


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111


0
0000
1
0001
2
0010
3
0011
4
0100
5
0101
6
0110
7
0111
-1
-2
-3
-4
-5
-6
-7
-8

Two's Complement

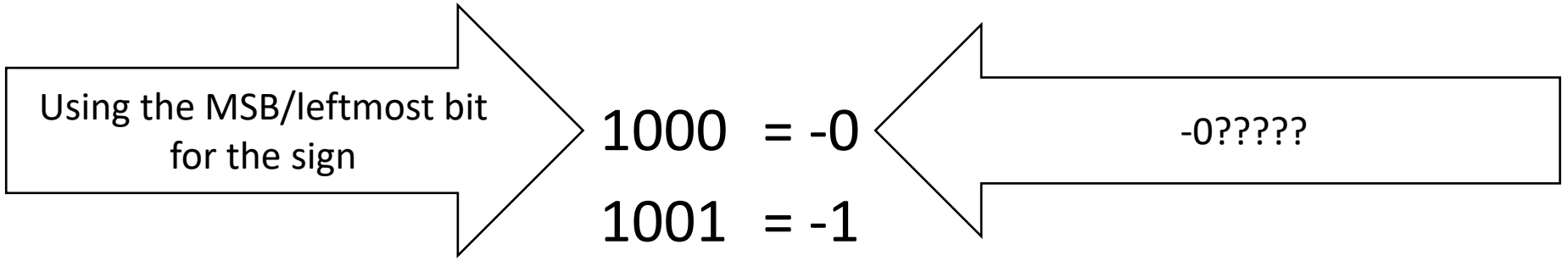
Positive 5 in binary

$0101_2 = \quad 2^0+2^2 = \quad 1+4 = 5$

So if we use the first bit to determine the sign, then negative 5 in binary would be

$1101_2 = -5$

Two's Complement



0000	= 0
0001	= 1
0010	= 2
0011	= 3
0100	= 4
0101	= 5
0110	= 6
0111	= 7

1000	= -0
1001	= -1
1010	= -2
1011	= -3
1100	= -4
1101	= -5
1110	= -6
1111	= -7

Two's Complement

0000 = 0	1000 = -0
0001 = 1	1001 = -1
0010 = 2	1010 = -2
0011 = 3	1011 = -3
0100 = 4	1100 = -4
0101 = 5	1101 = -5
0110 = 6	1110 = -6
0111 = 7	1111 = -7

Using MSB to hold the sign

```
    5          0101₂
+ (-5)       + 1101₂
------       ------
    0          0010₂
```

2_{10} ← 5+(-5) = 2????

```
    7          0111₂
+ (-3)       + 1011₂
------       ------
    4          0010₂
```

2_{10} ← 7+(-3) = 2????

```
    6          0110₂
+ (-2)       + 1010₂
------       ------
    4          0000₂
```

0_{10} ← 6+(-2) = 0????

Two's Complement

0000	= 0	1111	= -0
0001	= 1	1110	= -1
0010	= 2	1101	= -2
0011	= 3	1100	= -3
0100	= 4	1011	= -4
0101	= 5	1010	= -5
0110	= 6	1001	= -6
0111	= 7	1000	= -7

one's complement
invert all bits

```
    5          0101₂
+ (-5)       + 1010₂
------       ------
    0          1111₂
```

all off by 1 place

-0_{10} ← 5+(-5)=-0????

```
    7          0111₂
+ (-3)       + 1100₂
------       ------
    4          0011₂
```

3_{10} ← 7+(-3) = 3????

```
    6          0110₂
+ (-2)       + 1101₂
------       ------
    4          0011₂
```

3_{10} ← 6+(-2) = 3????

Two's Complement

So one's complement gets us close to representing a negative binary number

using the MSB for the sign

only off by 1

uses a -0 and +0

1111 = -0

1110 = -1

1101 = -2

1100 = -3

1011 = -4

1010 = -5

1001 = -6

1000 = -7

Two's Complement

One's complement

0000	= 0	1111	= -0
0001	= 1	1110	= -1
0010	= 2	1101	= -2
0011	= 3	1100	= -3
0100	= 4	1011	= -4
0101	= 5	1010	= -5
0110	= 6	1001	= -6
0111	= 7	1000	= -7

Two's Complement

0000	= 0	1111	= -1
0001	= 1	1110	= -2
0010	= 2	1101	= -3
0011	= 3	1100	= -4
0100	= 4	1011	= -5
0101	= 5	1010	= -6
0110	= 6	1001	= -7
0111	= 7	1000	= -8

Took out the -0

Added 1 to one's complement

Two's Complement

0000	= 0	1111	= -1
0001	= 1	1110	= -2
0010	= 2	1101	= -3
0011	= 3	1100	= -4
0100	= 4	1011	= -5
0101	= 5	1010	= -6
0110	= 6	1001	= -7
0111	= 7	1000	= -8

two's complement

```
    5          0101₂
+ (-5)       + 1011₂
------       ------
    0          0000₂        0₁₀
```

```
    7          0111₂
+ (-3)       + 1101₂
------       ------
    4          0100₂        4₁₀
```

```
    6          0110₂
+ (-2)       + 1110₂
------       ------
    4          0100₂        4₁₀
```

Two's Complement

How to calculate the two's complement of a number

0000 = 0	1111 = -1
0001 = 1	1110 = -2
0010 = 2	1101 = -3
0011 = 3	1100 = -4
0100 = 4	1011 = -5
0101 = 5	1010 = -6
0110 = 6	1001 = -7
0111 = 7	1000 = -8

Take

5_{10} which is 0101_2

invert it 1010_2

and add 1

$$\begin{array}{r} 1010_2 \\ +\quad 1 \\ \hline 1011_2 \end{array}$$

-5_{10}

Take

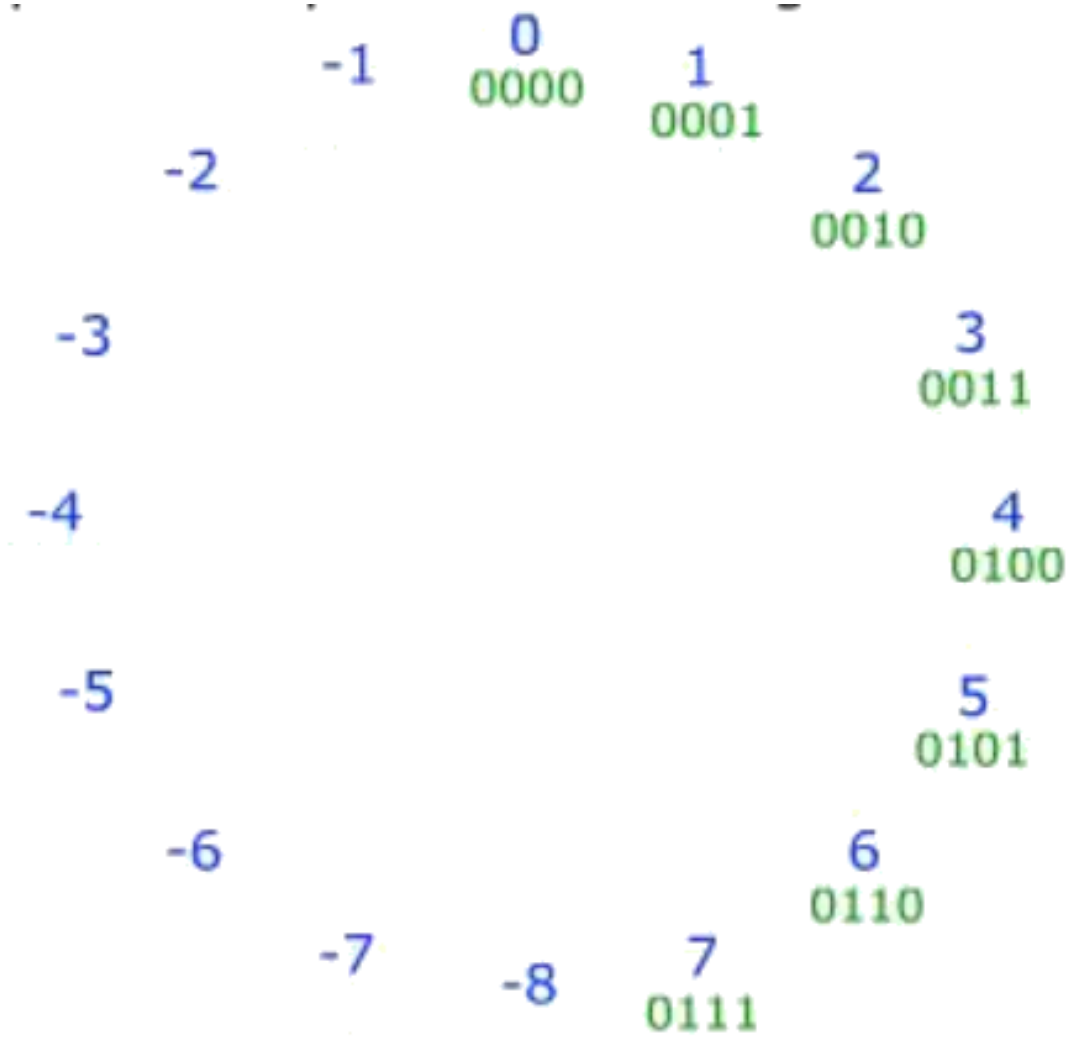
3_{10} which is 0011_2

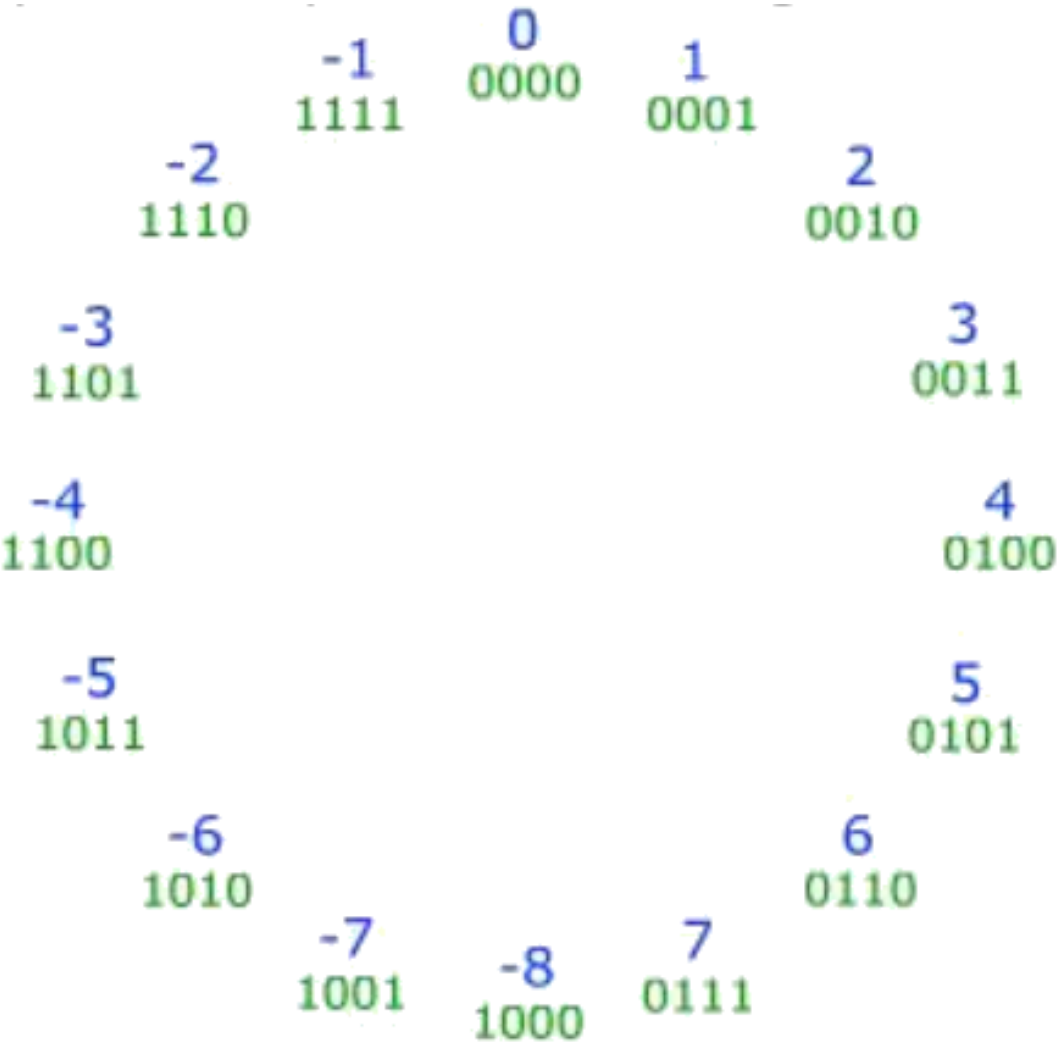
invert it 1100_2

and add 1

$$\begin{array}{r} 1100_2 \\ +\quad 1 \\ \hline 1101_2 \end{array}$$

-3_{10}


0
0000
1
0001
2
0010
3
0011
4
0100
5
0101
6
0110
7
0111
-1
-2
-3
-4
-5
-6
-7
-8


0
0000
1
0001
2
0010
3
0011
4
0100
5
0101
6
0110
7
0111
-1
1111
-2
1110
-3
1101
-4
1100
-5
1011
-6
1010
-7
1001
-8
1000

Function Definitions

function definition includes

- function type
- function name
- names, types and number of formal parameters
- executable statements for the function

```
type function_name(int param1, int param2)
{
   /* function code goes here */
}

void MyFunction(int DecNum)
{
   print("DecNum is %d\n", DecNum);
}
```

Function Types

default type for C is type `int`

- any other type must be explicitly declared

type of the function matches the type of the expression associated with its return value

void can be used to not return a value

- no return statement
- return without an expression

```c
int main(void)
{
  int addend1;
  int addend2;

  printf("Enter first  addend ");
  scanf("%d", &addend1);
  printf("\nEnter second addend ");
  scanf("%d", &addend2);

  PrintSum(addend1, addend2);

  return 0;
}
```

```c
void PrintSum(int Add1, int Add2)
{
  int a;

  printf("\n\t%5d\n", Add1);
  printf("\t\b+%5d\n\t", Add2);

  for (a = 0; a < 5; a++)
  {
    printf("=");
  }

  printf("\n\t%5d\n", Add1 + Add2);

  return;
}
```

function1Demo.c

```
int main(void)
{
  int addend1;
  int addend2;

  printf("Enter first  addend ");
  scanf("%d", &addend1);
  printf("\nEnter second addend ");
  scanf("%d", &addend2);

  printf("\n\t%5d\n", PrintSum(addend1, addend2));

  return 0;
}
```

```
int PrintSum(int Add1, int Add2)
{
  int a;

  printf("\n\t%5d\n", Add1);
  printf("\t\b+%5d\n\t", Add2);

  for (a = 0; a < 5; a++)
  {
    printf("=");
  }

  return Add1 + Add2;
}
```

function2Demo.c

Function Prototypes

A function prototype is a declaration of a function that tells the compiler the function's name, its return type and the types of its parameters.

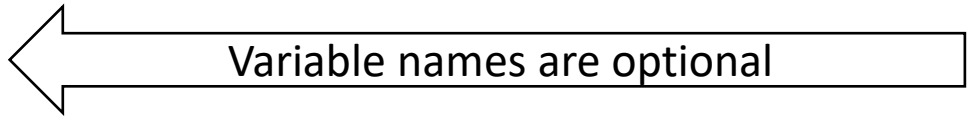
The function prototype is the *same* as the first line of the corresponding function definition, but ends with a *required* semicolon.

Function Definition

```
int PrintSum(short Add1, short Add2)
```

Function Prototype

```
int PrintSum(short Add1, short Add2);
int PrintSum(short, short);
```



Function Prototypes

The compiler uses the prototype to

- Ensure that the function definition matches the function prototype.
- Check that the function call contains the correct number and types of arguments and that the types of the arguments are in the correct order.
- Ensure that the value returned by the function can be used correctly in the expression that called the function—for example, for a function that returns void you cannot call the function on right side of an assignment.
- Ensure that each argument is consistent with the type of the corresponding parameter—for example, a parameter of type double can receive values like 7.35, 22 or –0.03456, but not a string like "hello".
- If the arguments passed to a function do not match the types specified in the function's prototype, the compiler attempts to convert the arguments to those types.

```
3  #include <stdio.h>
4
5  void CallA(void)
6  {
7  }
8
9  void CallB(void)
10 {
11 }
12
13 void CallC(void)
14 {
15 }
16
17 int main(void)
18 {
19     CallA();
20     CallB();
21     CallC();
22
23     return 0;
24 }
25
```

When the compiler runs…

Line 5 - Nice to meet you `CallA`
Line 9 - Nice to meet you `CallB`
Line 13 - Nice to meet you `CallC`

Line 19 - Hi `CallA` – we've met
Line 20 - Hi `CallB` – we've met
Line 21 - Hi `CallC` – we've met

Function Prototypes

What happens if we move `main()` to the top of the program but do not add prototypes?

```c
#include <stdio.h>

int main(void)
{
    CallA();
    CallB();
    CallC();

    return 0;
}

void CallA(void)
{
}

void CallB(void)
{
}

void CallC(void)
{
}
```

```
ProtoDemo.c:15: warning: conflicting types for ‘CallA’
ProtoDemo.c:7: warning: previous implicit declaration of
‘CallA’ was here

ProtoDemo.c:19: warning: conflicting types for ‘CallB’
ProtoDemo.c:8: warning: previous implicit declaration of
‘CallB’ was here

ProtoDemo.c:23: warning: conflicting types for ‘CallC’
ProtoDemo.c:9: warning: previous implicit declaration of
‘CallC’ was here
```

```
5  void CallA(void);
6  void CallB(void);
7  void CallC(void);
8
9  int main(void)
10 {
11     CallA();
12     CallB();
13     CallC();
14
15     return 0;
16 }
17
18 void CallA(void)
19 {
20 }
21
22 void CallB(void)
23 {
24 }
25
26 void CallC(void)
27 {
28 }
```

When the compiler runs…

Line 5 - Nice to meet you `CallA`
Line 6 - Nice to meet you `CallB`
Line 7 - Nice to meet you `CallC`

Line 11 - Hi `CallA` – we've met
Line 12 - Hi `CallB` – we've met
Line 13 - Hi `CallC` – we've met

```
 9  void CallA(void)
10  {
11      CallC();
12  }
13
14  void CallB(void)
15  {
16      CallC();
17  }
18
19  void CallC(void)
20  {
21      CallA();
22      CallB();
23  }
24
25  int main(void)
26  {
27      CallA();
28      CallB();
29      CallC();
30
31      return 0;
32  }
```

When the compiler runs…

Line 9 – Nice to meet you `CallA`
Line 11 – Nice to meet you `CallC`
Line 14 – Nice to meet you `CallB`
Line 16 – Hi `CallC` – we've met
Line 19 – Hi `CallC` – wait you weren't `void` before!?

```
[frenchdm@omega ~]$ gcc ProtoDemo.c -g
ProtoDemo.c:20: warning: conflicting types for
‘CallC’
ProtoDemo.c:11: warning: previous implicit
declaration of ‘CallC’ was here
[frenchdm@omega ~]$
```

? :

Known as

- Conditional operator
- inline if (iif)
- **ternary if**

```
if (condition)
{
    variable = expr1;
}
else
{
    variable = expr2;
}
```

```
variable = (condition) ? expr1 : expr2;
```

```
if (bit1 % 2)
{
    bit1 = 1;
}
else
{
    bit1 = 0;
}
```

```
bit1  = (bit1 % 2) ? 1 : 0;
```

? :

```
printf("Enter a number ");
scanf("%d", &MyNumber);

if (myNumber & 1)
{
   x = 1;
}
else
{
   y = 1;
}
```

Turn this into a ternary if

```
variable = condition ? expr1 : expr2;
```

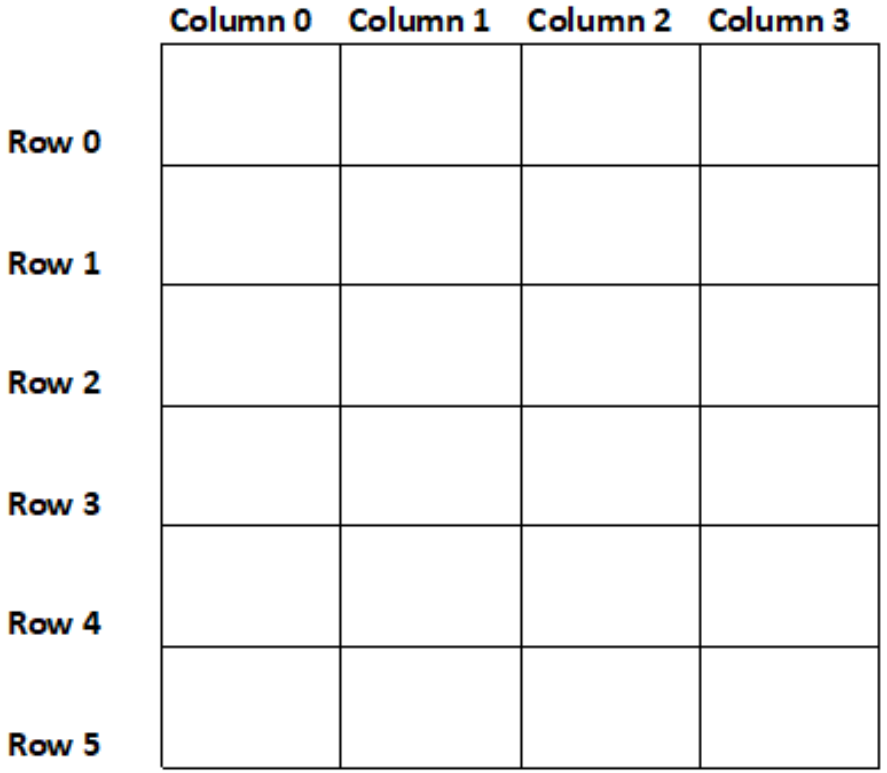
```
printf("Enter a number ");
scanf("%d", &MyNumber);

if (myNumber & 1)
{
   x = 1;
}
else
{
   x = 0;
}
```

Turn this into a ternary if

```
x = (myNumber & 1) ? 1 : 0;
```

Two-Dimensional Arrays

A two-dimensional array can be thought of as a matrix of elements.

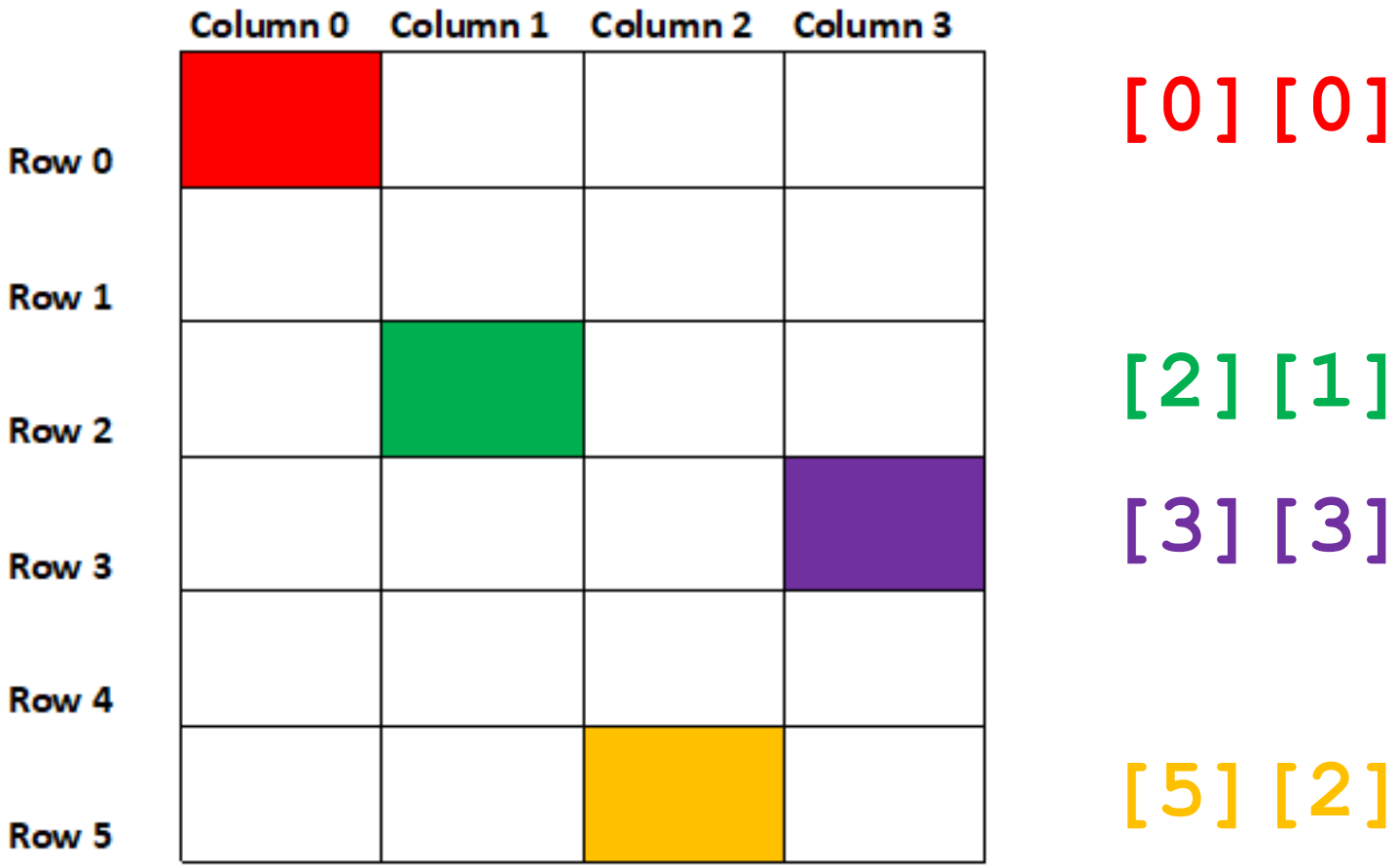


Two-Dimensional Arrays

Indices start at 0 and positions are referred to by row, column

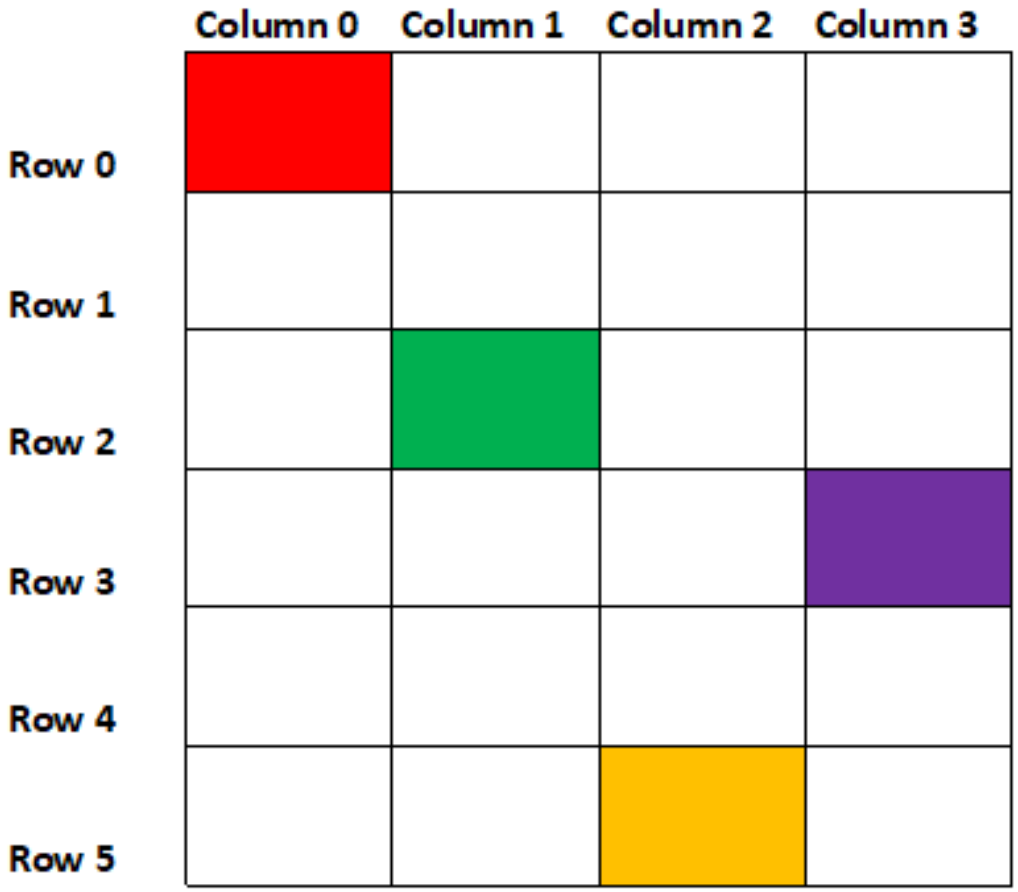
	Column 0	Column 1	Column 2	Column 3
Row 0				
Row 1				
Row 2				
Row 3				
Row 4				
Row 5				

Two-Dimensional Arrays



Two-Dimensional Arrays

```
int My2DArray[6][4];
```

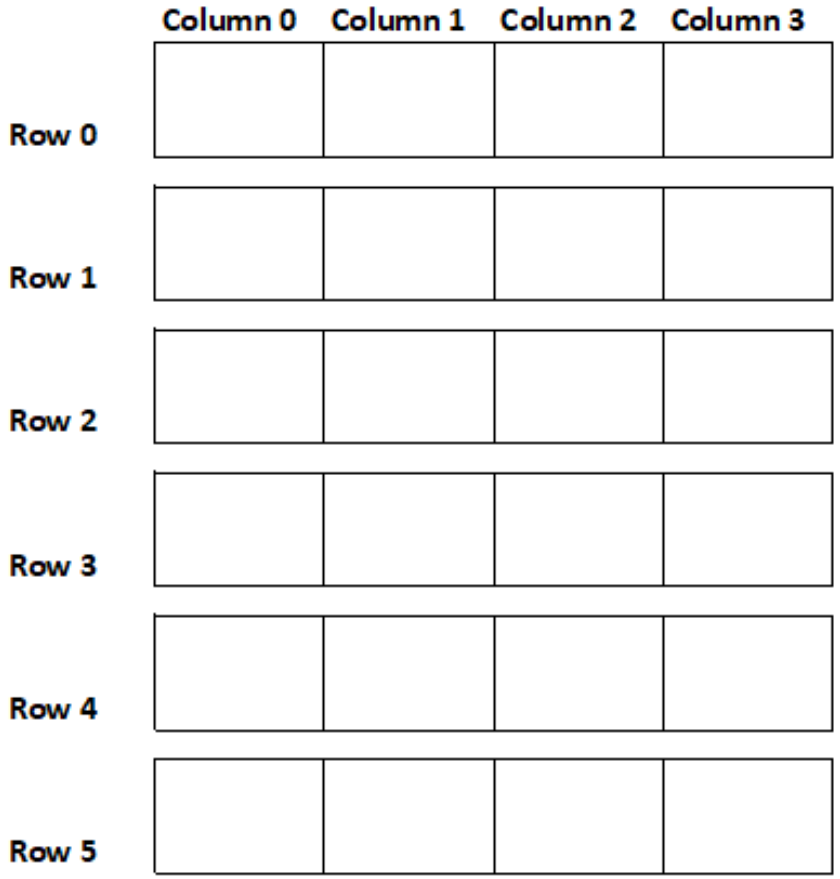



```
My2DArray[0][0];
My2DArray[2][1];
My2DArray[3][3];
My2DArray[5][2];
```

Two-Dimensional Arrays

Two-dimensional arrays are more accurately described as an array of arrays.

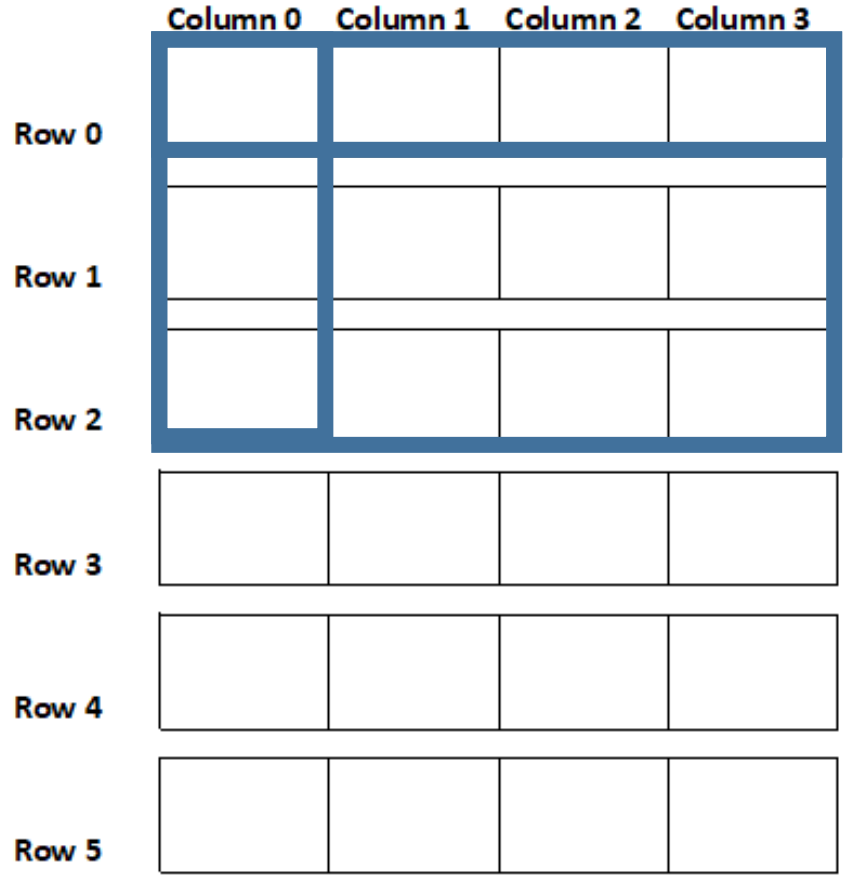
```
int (My2DArray[6])[4];
```



Multi-Dimensional Arrays

Multi-dimensional arrays are more accurately described as arrays of arrays.

```
int My2DArray[6][4];

int (My2DArray[6])[4];

int My2DArray[3][4];
int My2DArray[3][1];
int My2DArray[1][4];
```



Two-Dimensional Arrays

```
Printing out a multiplication table

        col0    col1    col2    col3    col4    col5    col6    col7    col8
-------------------------------------------------------------------------------
row0  |  1       2       3       4       5       6       7       8       9
row1  |  2       4       6       8       10      12      14      16      18
row2  |  3       6       9       12      15      18      21      24      27
row3  |  4       8       12      16      20      24      28      32      36
row4  |  5       10      15      20      25      30      35      40      45
row5  |  6       12      18      24      30      36      42      48      54
row6  |  7       14      21      28      35      42      49      56      63
row7  |  8       16      24      32      40      48      56      64      72
row8  |  9       18      27      36      45      54      63      72      81
```

```
// Declare a two dimensional array and initialize to NULLs
int MultTable[9][9] = {};

for (i = 1; i <= 9; i++)
{
   for (j = 1; j <= 9; j++)
   {
      MultTable[i-1][j-1] = i * j;
   }
}
```

```
{{0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0}}
```

MultTable[0][0] = 1 * 1;

```
{{1, 2, 3, 4, 5, 6, 7, 8, 9},
 {2, 4, 6, 8, 10, 12, 14, 16, 18},
 {3, 6, 9, 12, 15, 18, 21, 24, 27},
 {4, 8, 12, 16, 20, 24, 28, 32, 36},
 {5, 10, 15, 20, 25, 30, 35, 40, 45},
 {6, 12, 18, 24, 30, 36, 42, 48, 54},
 {7, 14, 21, 28, 35, 42, 49, 56, 63},
 {8, 16, 24, 32, 40, 48, 56, 64, 72},
 {9, 18, 27, 36, 45, 54, 63, 72, 81}}
```

```
/* print heading */
printf("\tcol0\tcol1\tcol2\tcol3\tcol4\tcol5\tcol6\tcol7\tcol8\n");

for (i = 0; i <= 75; i++)
   printf("-");

printf("\n");

for (i = 0, k = 1; i < 9; i++, k
{
   printf("row%d |\t", k-1);
   for (j = 0; j < 9; j++)
   {
      printf("%d\t", MultTable[i][j]);
   }
   printf("\n");
}
```

Print the dashes under the columns

Print "row x | " at start of each row

Print one row of table

Bring cursor back to start of line

	col0	col1	col2	col3	col4	col5	col6	col7	col8
row0 |	1	2	3	4	5	6	7	8	9
row1 |	2	4	6	8	10	12	14	16	18
row2 |	3	6	9	12	15	18	21	24	27
row3 |	4	8	12	16	20	24	28	32	36
row4 |	5	10	15	20	25	30	35	40	45
[illegible] |	6	12	18	24	30	36	42	48	54
[illegible] |	7	14	21	28	35	42	49	56	63
row7 |		16	24	32	40	48	56	64	72
row8 |		18	27	36	45	54	63	72	81

Arrays with More Than Two Dimensions

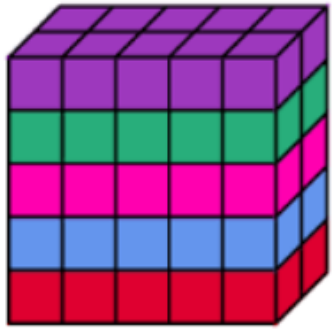
one dimension

two dimensions

array of 1D arrays

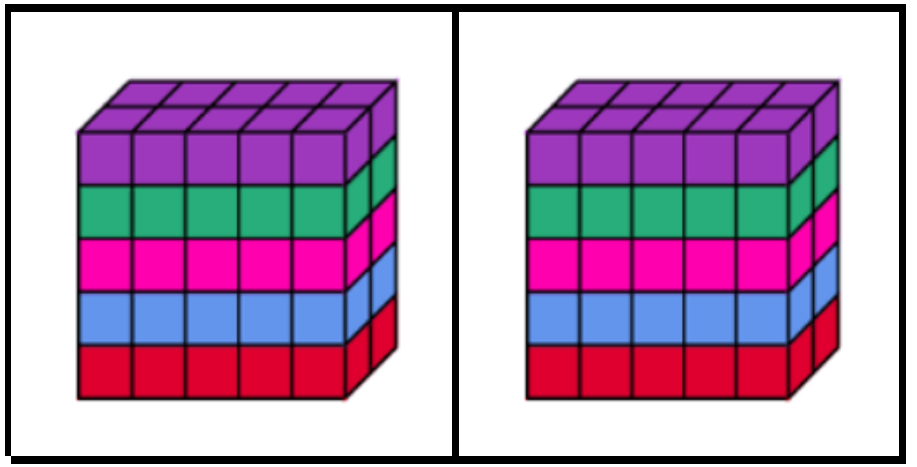
three dimensions

array of 2D arrays

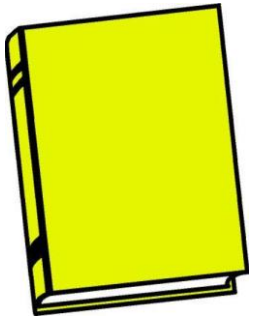


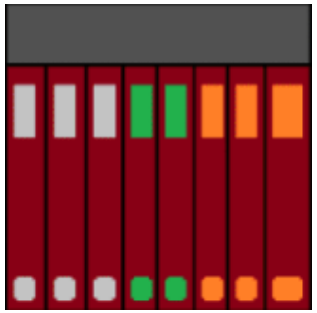
four dimensions

array of 3D arrays

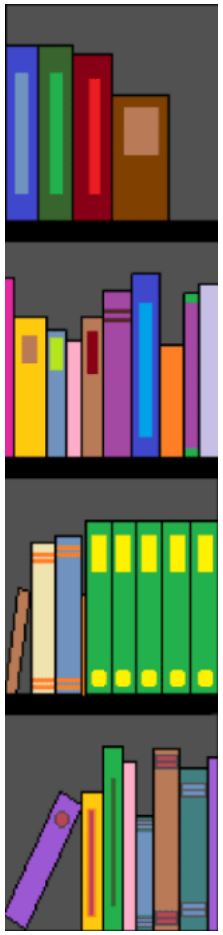


If a single book is an array of pages,
then a shelf of books is an array of arrays (2D array).





If a shelf of books is a 2D array,
then a bookcase is an array of 2D arrays (3D array).

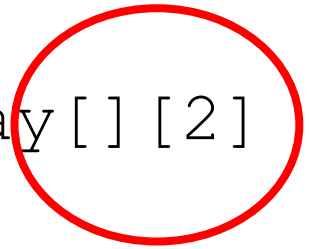


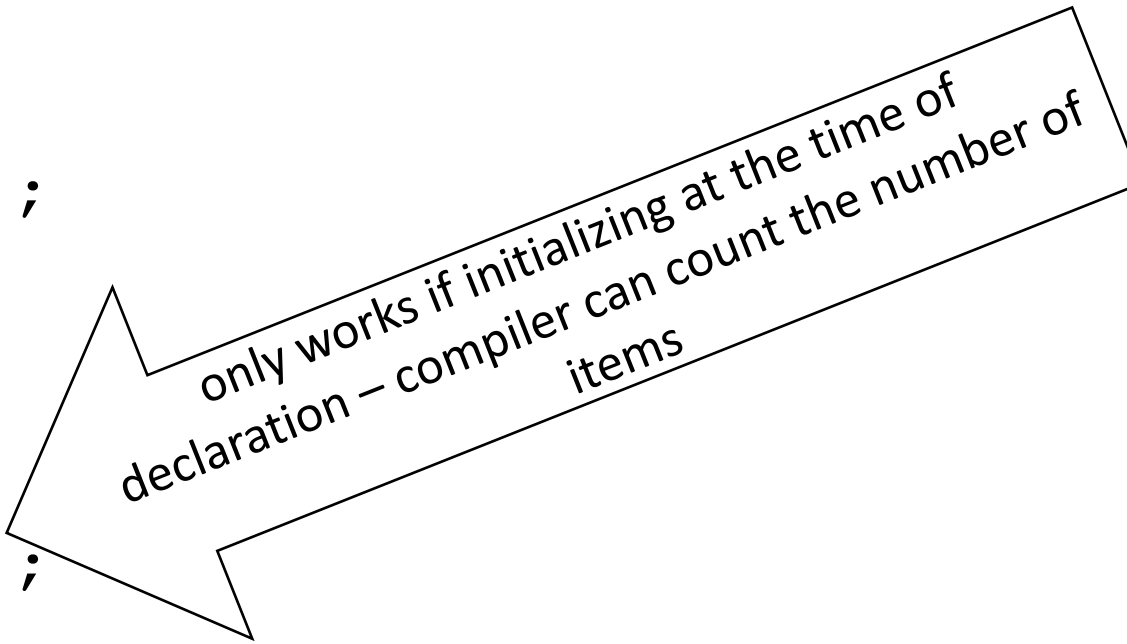
If a bookcase is a 3D array,
then a set of bookcases is an array of 3D arrays (4D array).

So how many dimensions could a library have?

Initializing Multidimensional Arrays

```
int My2DArray[3][2] = {};

int My2DArray[3][2] = {{1, 2},
                       {3, 4},
                       {5, 6}};

int My2DArray[][2]  = {{1, 2},
                       {3, 4},
                       {5, 6}};

char Alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

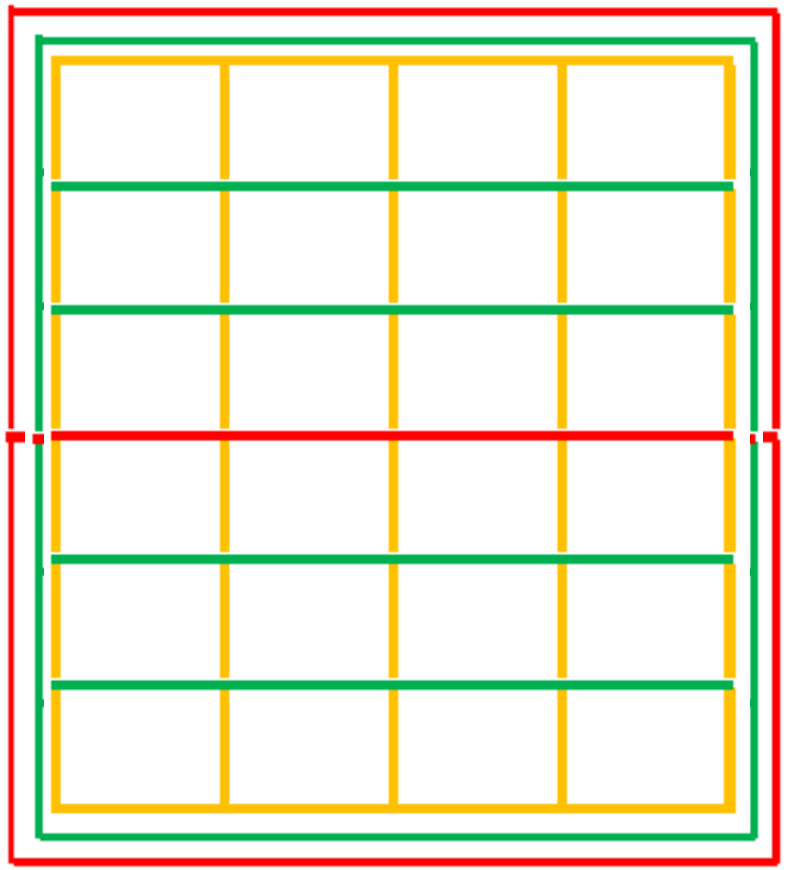




Arrays with More Than Two Dimensions

```
float My3DArray[2]          = {0, 1};

float My3DArray[2][3]       = {{0.0, 0.1, 0.2},
                               {1.0, 1.1, 1.2}};
```

Arrays with More Than Two Dimensions



```
float My3DArray[2][3][4] = {
    {  {0.00,0.01,0.02,0.03},
       {0.10,0.11,0.12,0.13},
       {0.20,0.21,0.22,0.23}
    },
    {  {1.00,1.01,1.02,1.03},
       {1.10,1.11,1.12,1.13},
       {1.20,1.21,1.22,1.23}
    },
};
```

Multidimensional Arrays as Parameters to Functions

When passing a one dimensional array to a function, we don't specify the number of elements in the function parameter because we are passing the address of the array.

```
void ConvertDecimalToBinary(int BinaryArray[])
```

The programmer needs to know how many elements are in the array in order to not go beyond the bounds of the array.

What if we wanted to move the creation of the multiplication table to its own function and the printing of the multiplication table to its own function? We could also allow the user to choose the size of the table.

```
int MultTable[MAX_ROW][MAX_COL] = {};
```

`MAX_ROW` and `MAX_COL` are both set to 9

```
printf("How many rows (1-9)? ");
scanf("%d", &row);
```

We don't deal with the <ENTER>. Why?

```
printf("How many columns (1-9)? ");
scanf("%d", &col);
```

```
FillOutMultTable(MultTable, row, col);

PrintMultTable(MultTable, row, col);
```

Passing the address of the array by using the name and the size of the array in `row` and `col`

`2d2Demo.c`

```
How many rows (1-9)? 5
```

```
printf("How many rows (1-9)? ");
scanf("%d", &row);
```

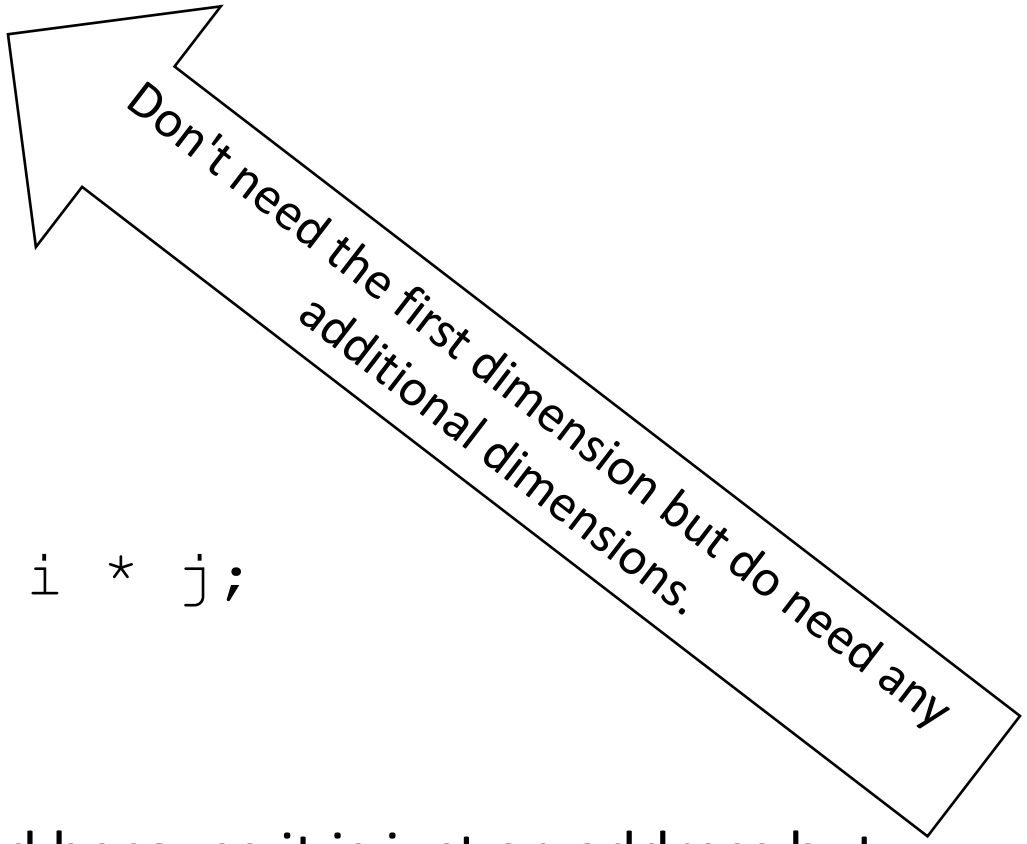
```
How many columns (1-9)? 5
```

```
printf("How many columns (1-9)? ");
scanf("%d", &col);
```

```
(gdb) p row
$7 = 5

(gdb) p col
$8 = 5
```

2d2Demo.c

```
void FillOutMultTable(int MultTable[][MAX_COL], int row, int col)
{
     int i, j;

     for (i = 1; i <= row; i++)
     {
          for (j = 1; j <= col; j++)
          {
               MultTable[i-1][j-1] = i * j;
          }
     }

     return;
}
```



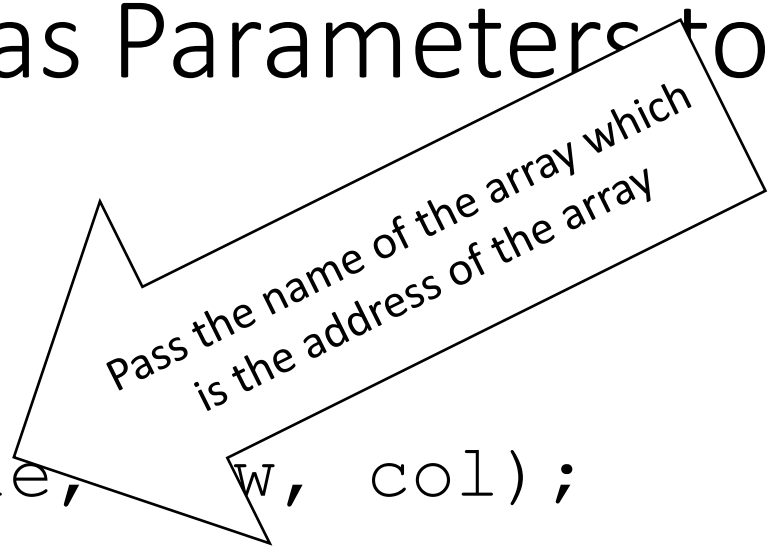
First dimension is not needed because it is just an address but when you add more dimensions, the compiler needs to know where each column starts and ends in order to properly divide up the contiguous memory reserved for the array.

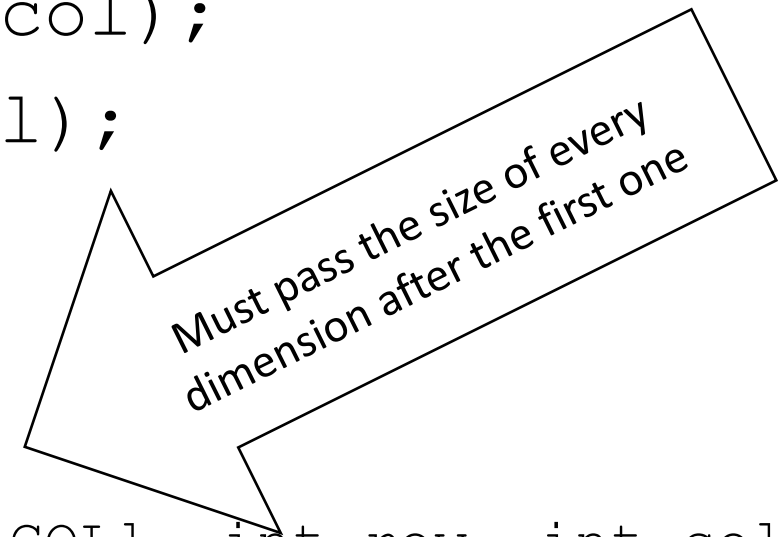
2d2Demo.c

Multidimensional Arrays as Parameters to Functions

Calling the functions

```
FillOutMultTable(MultTable,     w, col);
PrintMultTable(MultTable, row, col);
```





The functions themselves

```
void FillOutMultTable(int MultTable[][MAX_COL], int row, int col)
void PrintMultTable(int MultTable[][MAX_COL], int row, int col)
```