

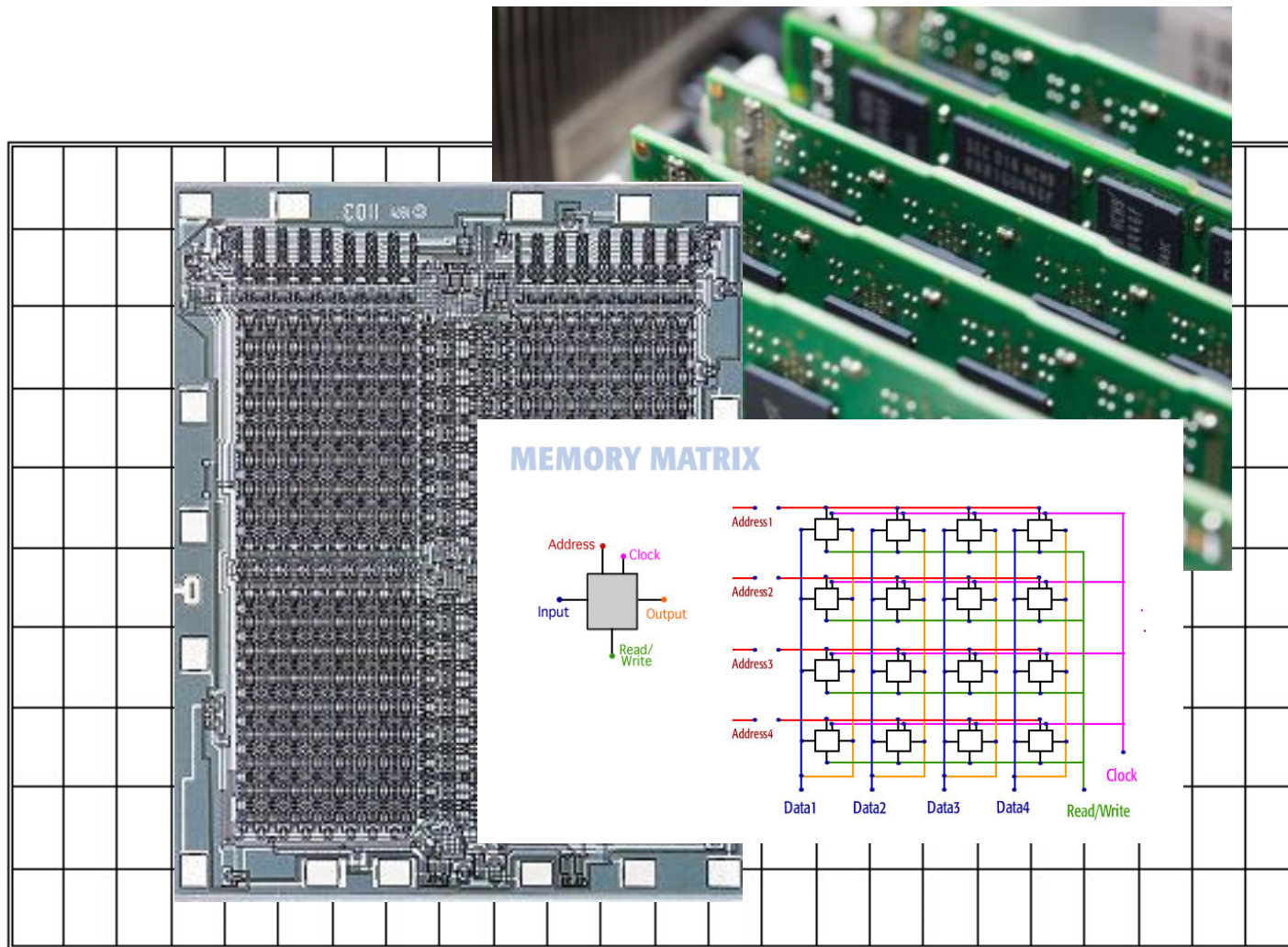
CS 2211

Systems Programming

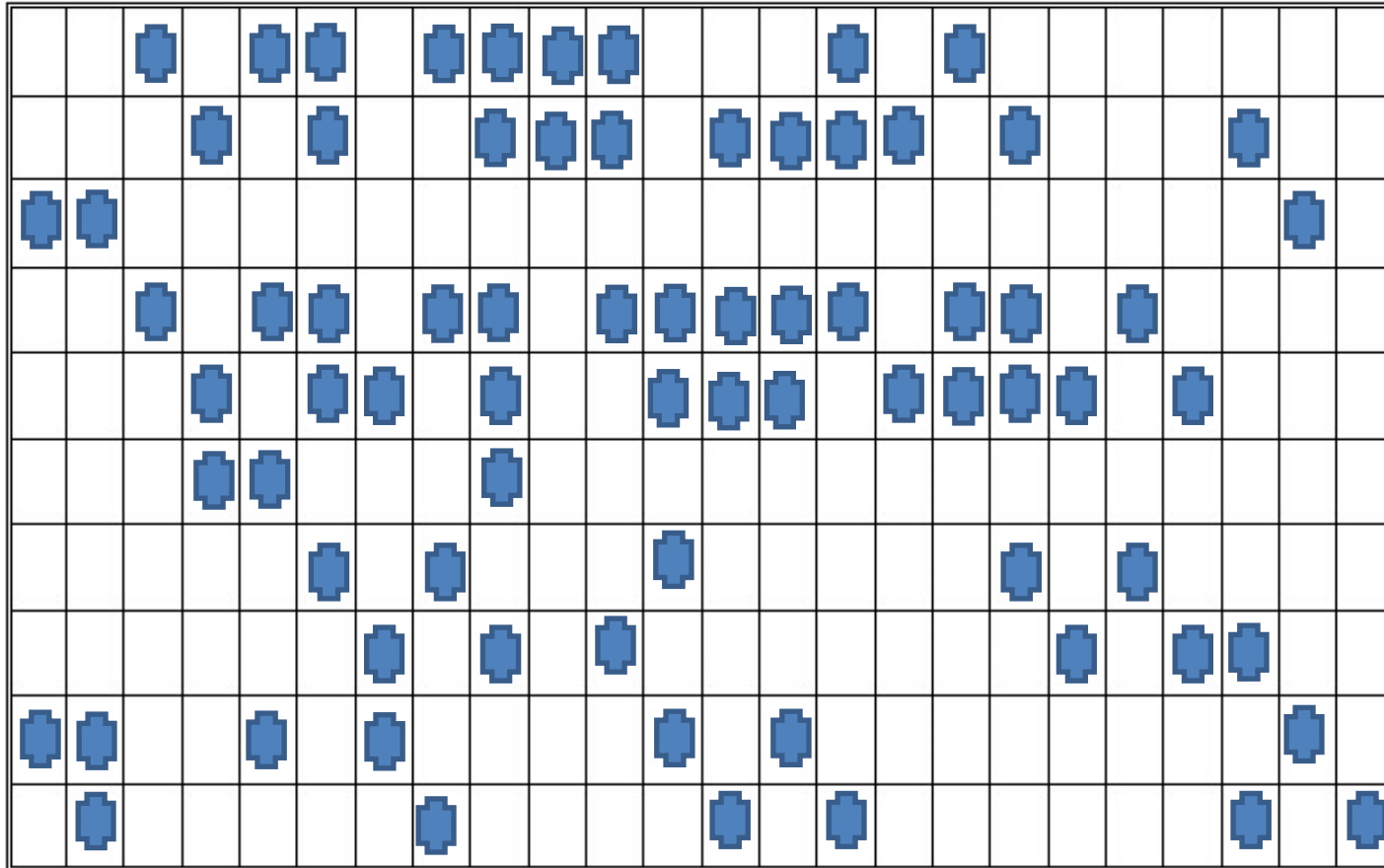
Part Four:

Memory Maps

COMPUTER MEMORY



COMPUTER MEMORY



COMPUTER MEMORY

1	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	1
0	0	1	0	0	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	1	1
0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1	0	1	0	0	0	1	1	0
1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	1	1
1	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	1
1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0
0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1	0	1	0	0	0	1	1	0
1	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	1
1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0

COMPUTER MEMORY

397 - 1 0 1 1 1 0 0 1	398 - 1 0 1 1 1 0 0 1	399 - 1 0 0 0 1 1 0 1
400 - 0 0 1 0 0 0 1 1	401 - 0 1 1 0 0 1 1 1	402 - 0 0 1 1 0 1 1 1
403 - 0 1 0 1 0 1 1 1	404 - 1 0 1 0 1 0 1 1	405 - 0 1 0 0 0 1 1 0
406 - 1 1 1 0 0 1 1 0	407 - 1 1 1 0 0 1 1 0	408 - 0 0 1 1 0 0 1 0
409 - 0 0 1 0 0 0 1 1	410 - 0 1 1 0 0 1 1 1	411 - 0 0 1 1 0 1 1 1
412 - 1 0 1 1 1 0 0 1	413 - 1 0 1 1 1 0 0 1	414 - 1 0 0 0 1 1 0 1
415 - 1 1 1 0 0 1 1 0	416 - 1 1 1 0 0 1 1 0	417 - 0 0 1 1 0 0 1 0
418 - 0 1 0 1 0 1 1 1	419 - 1 0 1 0 1 0 1 1	420 - 0 1 0 0 0 1 1 0
421 - 1 0 1 1 1 0 0 1	422 - 1 0 1 1 1 0 0 1	423 - 1 0 0 0 1 1 0 1
424 - 1 1 1 0 0 1 1 0	425 - 1 1 1 0 0 1 1 0	426 - 0 0 1 1 0 0 1 0

COMPUTER MEMORY

1	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	1
0	0	1	0	0	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	1	1
0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1	0	1	0	0	0	1	1	0
1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	1	1
1	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	1
1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0
0	1	0	1	0	1	1	1	1	0	1	0	1	0	1	1	0	1	0	0	0	1	1	0
1	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	1
1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0

COMPUTER MEMORY

397 -	398 -	399 -
1 0 1 1 1 0 0	1 1 0 1 1 1	0 0 1 1 0 0 0 1 1 0 1
400 -	401 -	402 -
0 0 1 0 0 0 1	1 0 1 1 0 0	1 1 1 0 0 1 1 0 1 1 1
403 -	404 -	405 -
0 1 0 1 0 1 1	1 1 0 1 0 1	0 1 1 0 1 0 0 0 1 1 0
406 -	407 -	408 -
1 1 1 0 0 1 1	0 1 1 1 0 0	1 1 0 0 0 1 1 0 0 1 0
409 -	410 -	411 -
0 0 1 0 0 0 1	1 0 1 1 0 0	1 1 1 0 0 1 1 0 1 1 1
412 -	413 -	414 -
1 0 1 1 1 0 0	1 1 0 1 1 1	0 0 1 1 0 0 0 1 1 0 1
415 -	416 -	417 -
1 1 1 0 0 1 1	0 1 1 1 0 0	1 1 0 0 0 1 1 0 0 1 0
418 -	419 -	420 -
0 1 0 1 0 1 1	1 1 0 1 0 1	0 1 1 0 1 0 0 0 1 1 0
421 -	422 -	423 -
1 0 1 1 1 0 0	1 1 0 1 1 1	0 0 1 1 0 0 0 1 1 0 1

MEMORY MAPPING

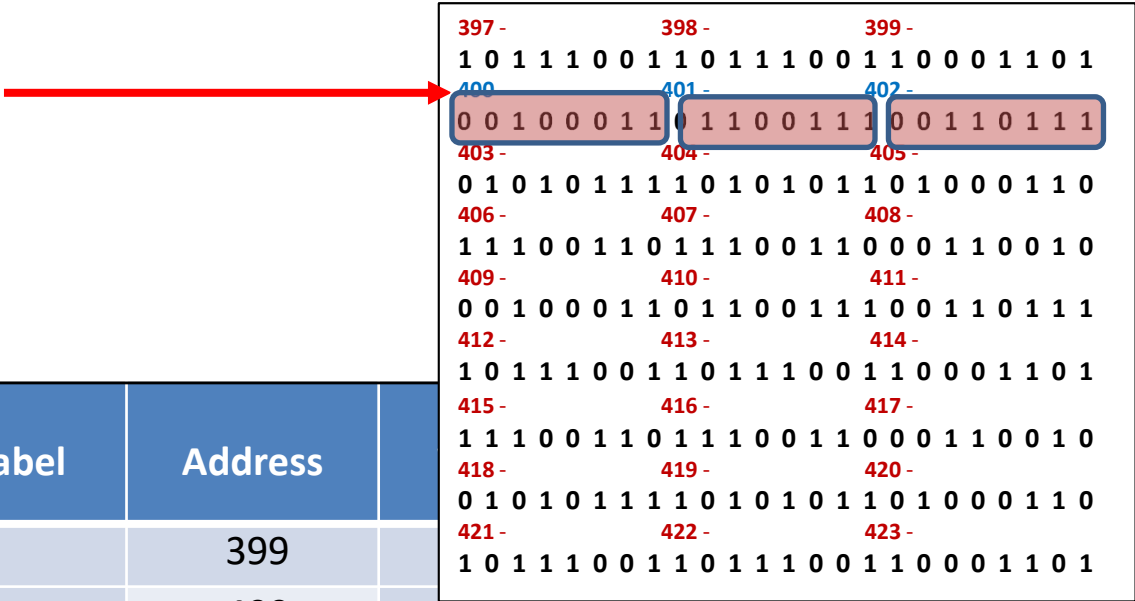
```
char a, b, c;
```

Type	Label	Address	Value	Binary
		399		
		400		
		401		
		402		
		403		
		404		
		405		
		406		
		...		

MEMORY MAPPING

```
char a, b, c; /* char - 1 byte each */
```

Type	Label	Address	
		399	
char	a	400	
char	b	401	
char	c	402	
		403	
		404	
		405	
		406	
		...	



MEMORY MAPPING

```
char a, b, c; /* char - 1 byte each */
```

Side bar:

actual computer addresses
reflect the size of the available
memory.

Actual addresses look like:

140727232603148
140727232603149
140727232603150

(-or displayed in hex:

0x7ffd9cb5a40c
0x7ffd9cb5a40d
0x7ffd9cb5a40e)

To make it easier to use and
understand, we will use
artificially small numbers like:
400 – 401 – 402 etc.

397 -	398 -	399 -
1 0 1 1 1 0 0 1	1 0 1 1 1 0 0 1	1 1 0 0 0 1 1 0 1
400 -	401 -	402 -
0 0 1 0 0 0 1 1	1 1 0 0 1 1 1	0 0 1 1 0 1 1 1
403 -	404 -	405 -
0 1 0 1 0 1 1 1	1 0 1 0 1 0 1 1	1 0 1 0 0 0 1 1 0
406 -	407 -	408 -
1 1 1 0 0 1 1 0	1 1 1 0 0 1 1 0	0 0 1 1 0 0 1 0
409 -	410 -	411 -
0 0 1 0 0 0 1 1	0 1 1 0 0 1 1 1	0 0 1 1 0 1 1 1
412 -	413 -	414 -
1 0 1 1 1 0 0 1	1 0 1 1 1 0 0 1	1 1 0 0 0 1 1 0 1
415 -	416 -	417 -
1 1 1 0 0 1 1 0	1 1 1 0 0 1 1 0	0 0 1 1 0 0 1 0
418 -	419 -	420 -
0 1 0 1 0 1 1 1	1 0 1 0 1 0 1 1	1 0 1 0 0 0 1 1 0
421 -	422 -	423 -
1 0 1 1 1 0 0 1	1 0 1 1 1 0 0 1	1 1 0 0 0 1 1 0 1

Address	
399	
400	
401	
402	
403	
404	
405	
406	
...	

MEMORY MAPPING

```
char a, b, c; /* char - 1 byte each */
```

note: not necessarily **contiguous**
assigned to locations that are 'free'
and available for the size required.

Type	Label	Address	
		399	
char	a	400	
		401	
		402	
char	c	403	
		404	
char	b	405	
		406	
		...	

397 -	398 -	399 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 1 1 0 1		
400 -	401 -	402 -
0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1	0 1 1 0 0 1 1	1 0 0 1 1 0 1 1
403 -	404 -	405 -
0 1 0 1 0 1 1 1	1 0 1 0 1 0 1	1 0 1 0 0 0 1 1 0
406 -	407 -	408 -
1 1 1 0 0 1 1 0	1 1 1 0 0 1 1	0 0 0 1 1 0 0 1 0
409 -	410 -	411 -
0 0 1 0 0 0 1 1	0 1 1 0 0 1 1	1 0 0 1 1 0 1 1 1
412 -	413 -	414 -
1 0 1 1 1 0 0 1	1 0 1 1 1 0 0	1 1 0 0 0 1 1 0 1
415 -	416 -	417 -
1 1 1 0 0 1 1 0	1 1 1 0 0 1 1	0 0 0 1 1 0 0 1 0
418 -	419 -	420 -
0 1 0 1 0 1 1 1	1 0 1 0 1 0 1	1 0 1 0 0 0 1 1 0
421 -	422 -	423 -
1 0 1 1 1 0 0 1	1 0 1 1 1 0 0	1 1 0 0 0 1 1 0 1

MEMORY MAPPING

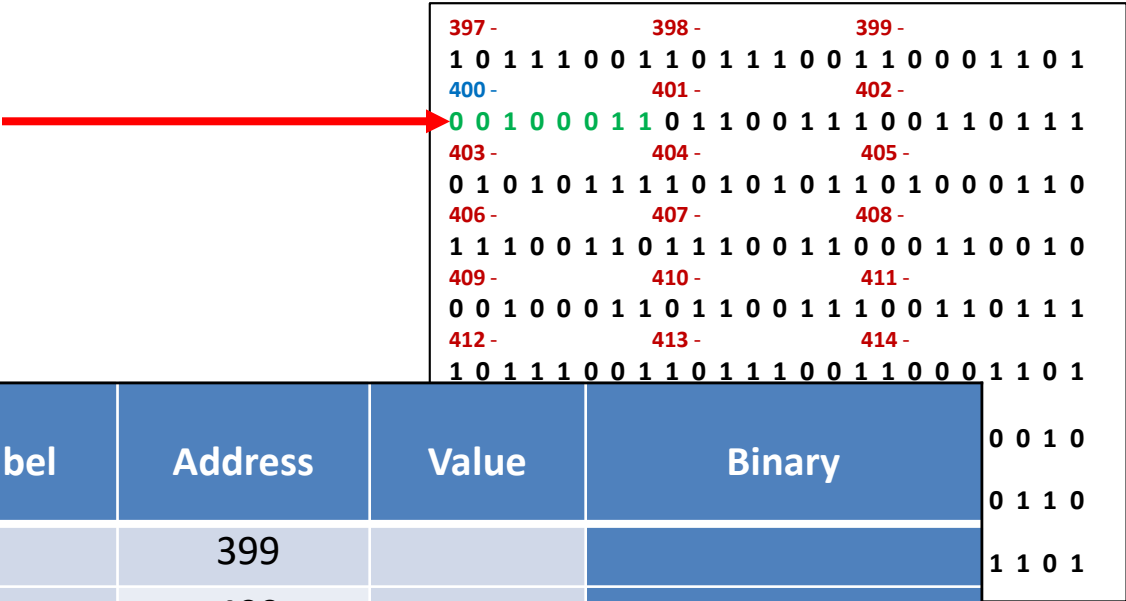
```
char a, b, c; /* char - 1 byte each */
```

397 - 398 - 399 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1
400 - 401 - 402 -
0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 1
403 - 404 - 405 -
0 1 0 1 0 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 0
406 - 407 - 408 -
1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0
409 - 410 - 411 -
0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 1
412 - 413 - 414 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1
415 - 416 - 417 -
1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 0 1 0
418 - 419 - 420 -
0 1 0 1 0 1 1 1 1 0 1 0 1 0 1 1 0 1 0 0 0 1 1 0
421 - 422 - 423 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1

Type	Label	Address		
		399		
char	a	400		
char	b	401		
char	c	402		
		403		
		404		
		405		
		406		
		...		

MEMORY MAPPING

```
char a, b, c; /* char - 1 byte each */
```



Type	Label	Address	Value	Binary
		399		
char	a	400		
char	b	401		
char	c	402		
		403		
		404		
		405		
		406		
		...		

MEMORY MAPPING

```
char a, b, c; /* char - 1 byte each */  
a = 7;        /* (0000 0111) */  
b = -13;      /* (1111 0011) */  
c = 0;        /* (0000 0000) */
```

Type	Label	Address	Value	Binary
		399		
char	a	400	7	0000 0111
char	b	401	-13	1111 0011
char	c	402	0	0000 0000
		403		
		404		
		405		
		406		
		...		

Memory Maps

END OF PART 1

MEMORY MAPPING

```
char a = 7;  
/* char: 1 byte */
```

Side bar:

char a = 7;

[variable declaration
- AND –
variable definition]

(computer process this as two
separate statements)

- 1.) char a;
2.) a = 7;

Type	Label	Address	Value	Binary
		399		
		400		
		401		
		402		
		403		
		404		
		405		
		406		
		407		
		408		
		409		
		410		
		411		
		412		
		413		
		414		
		415		
		416		
		417		
		418		
		...		

MEMORY MAPPING

```
char a = 7;  
/* char: 1 byte */
```

Side bar:

char a = 7;

[variable declaration
- AND -
variable definition]

(computer process this as two
separate statements)

- 1.) char a;
2.) a = 7;

Type	Label	Address	Value	Binary
		399		
		400		
		401		
		402		
		403		
		404		
		405		
		406		
		407		
		408		
		409		
		410		
		411		
		412		
		413		
		414		
		415		
		416		
		417		
		418		
		...		

MEMORY MAPPING

```
char a = 7;  
/* char: 1 byte */
```

Side bar:

char a = 7;

[variable declaration
- AND –
variable definition]

(computer process this as two
separate statements)

- 1.) char a;
2.) a = 7;

Type	Label	Address	Value	Binary
		399		
		400		
		401		
		402		
		403		
		404		
		405		
		406		
		407		
		408		
		409		
		410		
		411		
		412		
		413		
		414		
		415		
		416		
		417		
		418		
		...		

MEMORY MAPPING

```
char a = 7;  
/* char: 1 byte */  
  
int b = -13;  
/* int: 4 bytes */  
  
float c = 0.1;  
/* float: 4 bytes */  
  
double d = 42.5;  
/* double: 8 bytes */
```

Type	Label	Address	Value	Binary
		399		
char	a	400	7	0000 0111
int	b	401	-13	1111 1111
		402		1111 1111
		403		1111 1111
		404		1111 0011
float	c	405	0.1	0011 1110
		406		0000 0000
		407		0000 0000
		408		0000 0000
double	d	409	42.5	0100 0000
		410		0100 0101
		411		0100 0000
		412		0000 0000
		413		0000 0000
		414		0000 0000
		415		0000 0000
		416		0000 0000
		417		
		418		
		...		

MEMORY MAPPING

```
char a = 7;
/* char: 1 byte */

int b = -13;
/* int: 4 bytes */

float c = 0.1;
/* float: 4 bytes */

double d = 42.5;
/* double: 8 bytes */
```

Type	Label	Address	Value	Binary
		399		
char	a	400	7	0000 0111
int	b	401	-13	1111 1111
		402		1111 1111
		403		1111 1111
		404		1111 0011
float	c	405	0.1	0011 1110
		406		0000 0000
		407		0000 0000
		408		0000 0000
double	d	409	42.5	0100 0000
		410		0100 0101
		411		0100 0000
		412		0000 0000
		413		0000 0000
		414		0000 0000
		415		0000 0000
		416		0000 0000
		417		
		418		
		...		

MEMORY MAPPING

```
char a;  
int b;  
float c;  
double d;
```

```
a = 7;          /* 1 byte */  
b = -13;        /* 4 bytes */  
c = 0.1;        /* 4 bytes */  
d = 42.5;       /* 8 bytes */
```

A BETTER WAY OF VISUALIZING THE MAP:

Label	Address	Value
	399	
a	400	7
b	401 - 404	-13
c	405 - 408	0.1
d	409 - 416	42.5
	417	
	418	
	419	
	...	

Memory Maps

END OF PART 2

MEMORY MAPPING

```
char a;  
int b;  
float c;  
double d;
```

ALLOCATION NOT NECESSARILY CONTIGUOUS:

Label	Address	Value
	399	
a	400	
	...	
d	409 - 416	
	...	
b	510 - 513	
	...	
c	605 - 608	
	...	
	...	
	...	
	...	

MEMORY MAPPING

```
char a;  
int b;  
float c;  
double d;
```

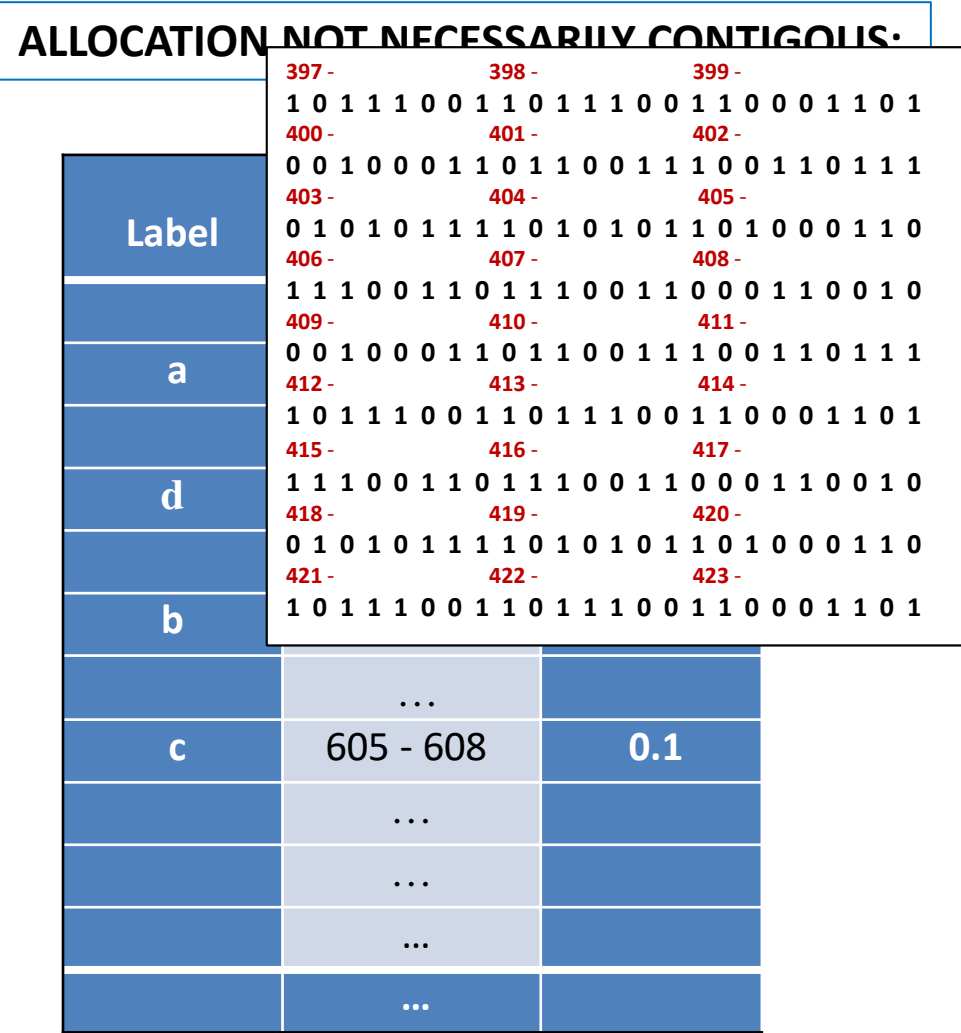
```
a = 7;          /* 1 byte */  
b = -13;        /* 4 bytes */  
c = 0.1;        /* 4 bytes */  
d = 42.5;       /* 8 bytes */
```

ALLOCATION NOT NECESSARILY CONTIGUOUS:

Label	Address	Value
	399	
a	400	7
	...	
d	409 - 416	42.5
	...	
b	510 - 513	-13
	...	
c	605 - 608	0.1
	...	
	...	
	...	
	...	

MEMORY MAPPING

```
char a;  
int b;  
float c;  
double d;  
  
a = 7;           /* 1 byte */  
b = -13;         /* 4 bytes */  
c = 0.1;         /* 4 bytes */  
d = 42.5;        /* 8 bytes */
```



MEMORY MAPPING

```
char a;  
int b;  
float c;  
double d;
```

VARIABLE DECLARATION VS. VARIABLE DEFINITION:

VARIABLE DECLARATION:

the variable is only declared
and allocated a block of memory
but still has no value

Label	Address	Value
	399	
a	400	
	...	
d	409 - 416	
	...	
b	510 - 513	
	...	
c	605 - 608	
	...	
	...	
	...	
	...	

MEMORY MAPPING

VARIABLE DECLARATION VS. VARIABLE DEFINITION:

```
a = 7;          /* 1 byte */
b = -13;        /* 4 bytes */
c = 0.1;        /* 4 bytes */
d = 42.5;       /* 8 bytes */
```

VARIABLE DEFINITION:

to assign or initialize it with
some specific value

Label	Address	Value
	399	
a	400	7
	...	
d	409 - 416	42.5
	...	
b	510 - 513	-13
	...	
c	605 - 608	0.1
	...	
	...	
	...	
	...	

Memory Maps

END OF PART 3

MEMORY MAPPING

Scalar Variables versus Aggregate Variables

So far, the only variables we've seen are **scalar**: capable of holding a single data item.

C also supports **aggregate** variables, which can store collections of values.

There are two kinds of aggregates in C: **arrays** and **structures**.

We will start by looking at one-dimensional arrays, which play a much bigger role in C than do multidimensional arrays.

An array is a data structure containing a number of data values, all of which have the same type.

These values, known as elements, can be individually selected by their position within the array.

The elements of a one-dimensional array are conceptually arranged one after another in a single row (or column):

MEMORY MAPPING

Scalar Variables versus Aggregate Variables

To declare an array, we must specify the type of the array's elements and the number of elements:

```
int a[10];
```

The elements may be of any type; the length of the array can be any (integer) constant expression.

An array, like any other variable, can be given an initial value at the time it's declared.

The most common form of array initializer is a list of constant expressions enclosed in braces and separated by commas:

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

If the initializer is shorter than the array, the remaining elements of the array are given the value 0:

```
int a[10] = {1, 2, 3, 4, 5, 6};  
/* initial value of a is {1, 2, 3, 4, 5, 6, 0, 0, 0, 0} */
```

If an initializer is present, the length of the array may be omitted:

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

The compiler uses the length of the initializer to determine how long the array is.

MEMORY MAPPING [ARRAYS]

```
int a[2];  
/* int: 4 bytes */  
  
float b[3];  
/* float: 4 bytes */  
  
double c[4];  
/* double: 8 bytes */  
  
char d[6];  
/* char: 1 byte */
```

Label	Address	Value	
a[0]	400 - 403		int a[2];
a[1]	404 - 407		
b[0]	408 - 411		float b[3];
b[1]	412 - 415		
b[2]	416 - 419		
c[0]	420 - 427		double c[4];
c[1]	428 - 435		
c[2]	436 - 443		
c[3]	444 - 451		
d[0]	452		char d[6];
d[1]	453		
d[2]	454		
d[3]	455		
d[4]	456		
d[5]	457		

MEMORY MAPPING [ARRAYS]

```
int a[2] = {37, 52};  
/* int: 4 bytes */
```

```
float b[3] = {26, 54};  
/* float: 4 bytes */
```

```
double c[4];  
/* double: 8 bytes */
```

```
char d[6];  
/* char: 1 byte */
```

```
c[2] = 14.7;
```

Label	Address	Value	
a[0]	400 - 403	37	int a[2];
a[1]	404 - 407	52	
b[0]	408 - 411	26	float b[3];
b[1]	412 - 415	54	
b[2]	416 - 419	0	double c[4];
c[0]	420 - 427	?	
c[1]	428 - 435	?	
c[2]	436 - 443	14.7	
c[3]	444 - 451	?	char d[6];
d[0]	452	?	
d[1]	453	?	
d[2]	454	?	
d[3]	455	?	
d[4]	456	?	
d[5]	457	?	
	...		

MEMORY MAPPING [ARRAYS]

```
int a[2] = {37, 52};
/* int: 4 bytes */
```

```
float b[3] = {26, 54};
/* float: 4 bytes */
```

```
double c[4];
/* double: 8 bytes */
```

```
char d[6];
/* char: 1 byte */
```

```
c[2] = 14.7;
```

```
d[4] = 'a';
```

```
/* BUT !now what happens if:
(because - this will compile
and run) */
```

```
b[4] = 15.9;
/* memory location 424-427 */
```

Label	Address	Value	
a[0]	400 - 403	37	int a[2];
a[1]	404 - 407	52	
b[0]	408 - 411	26	float b[3];
b[1]	412 - 415	54	
b[2]	416 - 419	0	double c[4];
c[0]	420 424 - 427	? 15.9	
c[1]	428 - 435	?	
c[2]	436 - 443	14.7	
c[3]	444 - 451	?	char d[6];
d[0]	452	?	
d[1]	453	?	
d[2]	454	?	
d[3]	455	?	
d[4]	456	a	
d[5]	457	?	
	...		

MEMORY MAPPING [ARRAYS]

```
int a[2] = {37, 52};
/* int: 4 bytes */

float b[3] = {26, 54};
/* float: 4 bytes */

double c[4];
/* double: 8 bytes */

char d[6];
/* char: 1 byte */

c[2] = 14.7;

d[4] = 'a';

/* OR !! This */

b[3333] = 15.9;
/* memory location 15205 */
run time error :
out of program allowed bounds
```

Label	Address	Value	
a[0]	400 - 403	37	int a[2];
a[1]	404 - 407	52	
b[0]	408 - 411	26	float b[3];
b[1]	412 - 415	54	
b[2]	416 - 419	0	double c[4];
c[0]	420 - 427	?	
c[1]	428 - 435	?	
c[2]	436 - 443	14.7	
c[3]	444 - 451	?	char d[6];
d[0]	452	?	
d[1]	453	?	
d[2]	454	?	
d[3]	455	?	
d[4]	456	a	
d[5]	457	?	
b[3333]	15205	15.9	

MEMORY MAPPING [ARRAYS]

**If an initializer is present,
the length of the array may be omitted:**

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

The compiler uses the length of the initializer to determine how long the array is.

[illegible]

Memory Maps

END OF PART 4

MEMORY MAPPING [ARRAYS – MULTIDIMENSIONAL]

```
int a[3][2];  
/* a 3 by 2 table */  
/* 3 rows and 2 columns */  
/* 6 cells of 4 bytes each */  
/* caution: never a[3,2] */
```

	0	1
0	0 0	0 1
1	1 0	1 1
2	2 0	2 1

Label	Address	Value
a[0][0]	400 - 403	
a[0][1]	404 - 407	
a[1][0]	408 - 411	
a[1][1]	412 - 415	
a[2][0]	416 - 419	
a[2][1]	420 - 423	

C stores arrays in **row-major order**, with row 0 first, then row 1, and so forth.
(sort of)

MEMORY MAPPING [ARRAYS – MULTIDIMENSIONAL]

```
int a[3][2];  
/* a 3 by 2 table */
```

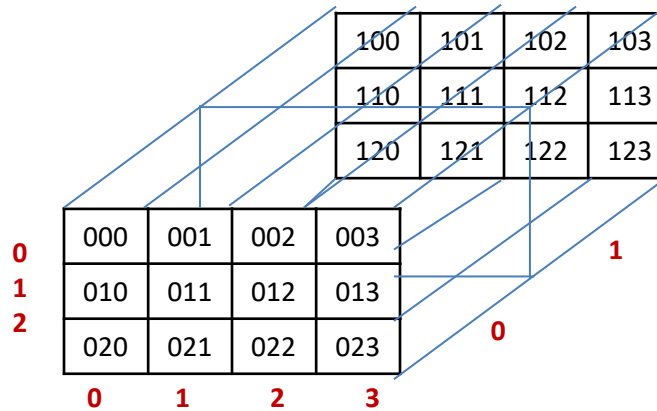
```
a[0][1] = 7;  
a[1][0] = 13;
```

	0	1
0		(7)
1	(13)	
2		

Label	Address	Value
a[0][0]	400 - 403	
a[0][1]	404 - 407	7
a[1][0]	408 - 411	13
a[1][1]	412 - 415	
a[2][0]	416 - 419	
a[2][1]	420 - 423	

MEMORY MAPPING [ARRAYS – THREE DIMENSIONAL]

```
int b[2][3][4];
/* a 2 by 3 by 4 table */
/* 2 layers 3 rows 4 columns */
/* 24 cells of 4 bytes each */
```

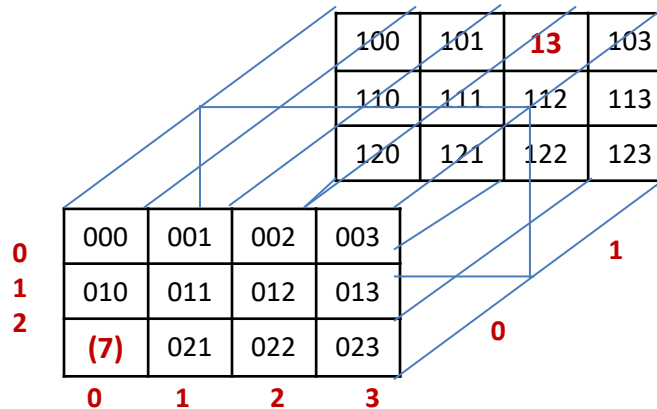


Label	Address	Value		Label	Address	Value
b[0][0][0]	400 - 403			b[1][0][0]	448 - 451	
b[0][0][1]	404 - 407			b[1][0][1]	452 - 455	
b[0][0][2]	408 - 411			b[1][0][2]	456 - 459	
b[0][0][3]	412 - 415			b[1][0][3]	460 - 463	
b[0][1][0]	416 - 419			b[1][1][0]	464 - 467	
b[0][1][1]	420 - 423			b[1][1][1]	468 - 471	
b[0][1][2]	424 - 427			b[1][1][2]	472 - 475	
b[0][1][3]	428 - 431			b[1][1][3]	476 - 479	
b[0][2][0]	432 - 435			b[1][2][0]	480 - 483	
b[0][2][1]	436 - 439			b[1][2][1]	484 - 485	
b[0][2][2]	440 - 443			b[1][2][2]	488 - 491	
b[0][2][3]	444 - 447			b[1][2][3]	492 - 495	

MEMORY MAPPING [ARRAYS – THREE DIMENSIONAL]

```
int b[2][3][4];
/* a 2 by 3 by 4 table */
```

```
b[0][2][0] = 7;
b[1][0][2] = 13;
```



Label	Address	Value	Label	Address	Value
<code>b[0][0][0]</code>	400 - 403		<code>b[1][0][0]</code>	448 - 451	
<code>b[0][0][1]</code>	404 - 407		<code>b[1][0][1]</code>	452 - 455	
<code>b[0][0][2]</code>	408 - 411		<code>b[1][0][2]</code>	456 - 459	13
<code>b[0][0][3]</code>	412 - 415		<code>b[1][0][3]</code>	460 - 463	
<code>b[0][1][0]</code>	416 - 419		<code>b[1][1][0]</code>	464 - 467	
<code>b[0][1][1]</code>	420 - 423		<code>b[1][1][1]</code>	468 - 471	
<code>b[0][1][2]</code>	424 - 427		<code>b[1][1][2]</code>	472 - 475	
<code>b[0][1][3]</code>	428 - 431		<code>b[1][1][3]</code>	476 - 479	
<code>b[0][2][0]</code>	432 - 435	7	<code>b[1][2][0]</code>	480 - 483	
<code>b[0][2][1]</code>	436 - 439		<code>b[1][2][1]</code>	484 - 487	
<code>b[0][2][2]</code>	440 - 443		<code>b[1][2][2]</code>	488 - 491	
<code>b[0][2][3]</code>	444 - 447		<code>b[1][2][3]</code>	492 - 495	

MEMORY MAPPING [VARIABLE LENGTH ARRAYS]

```
int n = 3;  
int a[n+1];  
/* int: 4 bytes */
```

ARRAY SIZE DETERMINED AT RUN TIME

Label	Address	Value
a[0]	400 - 403	
a[1]	404 - 407	
a[2]	408 - 411	
a[3]	412 - 415	

int a[n+1];

Memory Maps

END OF PART 5

MEMORY MAPPING [STRINGS – A SPECIAL ARRAY OF CHARACTERS]

```
char d[8]
/* char: 1 byte */
```

```
d[0] = 'H'
d[1] = 'e';
d[2] = 'l';
d[3] = 'l';
d[4] = 'o';
d[5] = '\\0';
```

ARRAY OF CHARACTERS IS NOT A STRING
-> until '\\0' is assigned (the NULL character)

Label	Address	Value	char d[8];
d[0]	400	H	
d[1]	401	e	
d[2]	402	l	
d[3]	403	l	
d[4]	404	o	
d[5]	405	\\0	
d[6]	406		
d[7]	407		

STRINGS:
(array of char : terminated with null)

MEMORY MAPPING [STRINGS – A SPECIAL ARRAY OF CHARACTERS]

STRING ARRAY assigned a VALUE

```
char d[8] = "Magic";  
/* char: 1 byte */
```

STRINGS:

(when assigned within quotes
automatically terminated with null)

STRINGS (SIZE):

(the size of the string at d is: 5
- not 8.
MORE ON THIS LATER...)

Label	Address	Value
d[0]	400	M
d[1]	401	a
d[2]	402	g
d[3]	403	i
d[4]	404	c
d[5]	405	\0
d[6]	406	0
d[7]	407	0

char d[8];

MEMORY MAPPING [STRINGS – A SPECIAL ARRAY OF CHARACTERS]

```
char d[] = "Code";  
/* char: 1 byte */
```

STRING ARRAY assigned a VALUE

```
char d[8];
```

STRINGS:

(when no size specified – array of characters automatically assigned just enough memory for characters PLUS the terminating NULL)

STRINGS (SIZE):

(the size of the string at d is: 5
MORE ON THIS LATER...)

[illegible]

Memory Maps

END OF PART 6

Use of a memory map in a dynamic way:

(tracing a program) - what's wrong with this ??

```
int i, sum;      not initialized sum  
int n = 0;  
for ( i=1; i <=10; i++)  
    if ( i % 2 == 0 )  
        sum = sum + 1 + i;
```

Label	Address	Value
	...	

Memory Maps

END OF PART 7