COMP 206 – Introduction to Software Systems

Lecture 4 – C Basics Continued

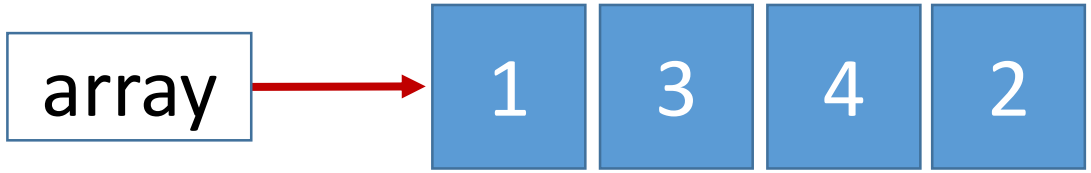
January 18th, 2018

Exercise: A programming challenge

- Using what we saw so far, can you:
  - Start with the array shown
  - Write code to sort the array in increasing order
  - Output the sorted list on the terminal

- Helper: linear sort specification
  - For each position in the array, pos
    - Let curr be the current value at pos
    - Find the position of the minimal element from pos to the end: min_val, min_pos
    - Swap the values of pos and min_pos

```
int array[4];
array[0] = 1;
array[1] = 3;
array[2] = 4;
array[3] = 2;
```



- Could you do this in the language you know?
- Try to translate each part into C
- Ask if you get stuck

Programming Challenge Solution

- Take a list of numbers, as command-line arguments

```
int main( int argc, char *argv[] )
int num_args = argc-1;
printf( “first arg is %s.\n”, argv[0] );
```

Programming Challenge Solution

- Store those numbers in an array

```
int array[num_args];
for(int pos=0; pos<num_args; pos++){
    array[pos] = atoi(argv[pos+1]));
}
```

Programming Challenge Solution

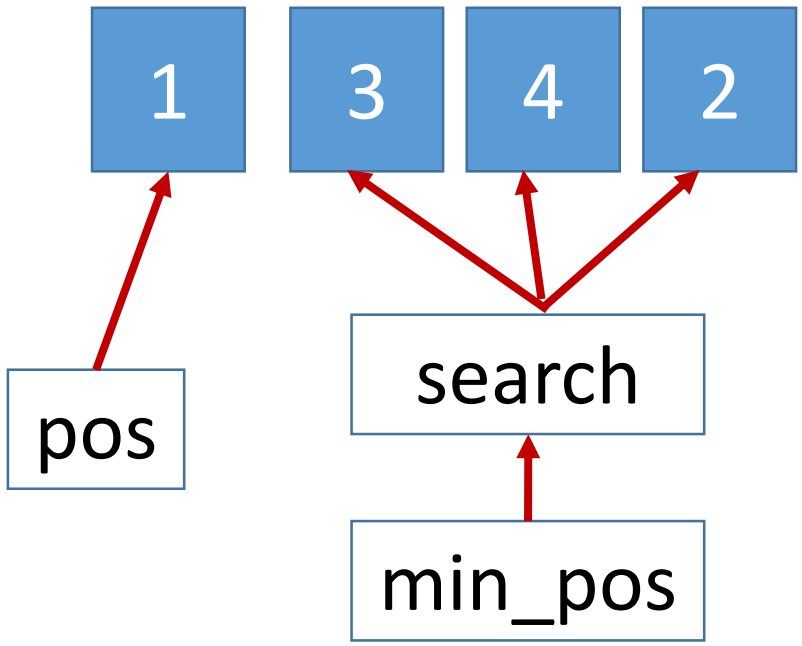
- Output the sorted list to terminal

```
for( int pos=0; pos<num_args; pos++){
    printf( “Element %d is %d.\n”, pos, array[pos] );
}
```

Sorting

- For each position in the array, pos
  - Let curr be the current value at pos
  - Find the position of the minimal element from pos to the end: min_val, min_pos
  - Swap the values of pos and min_pos

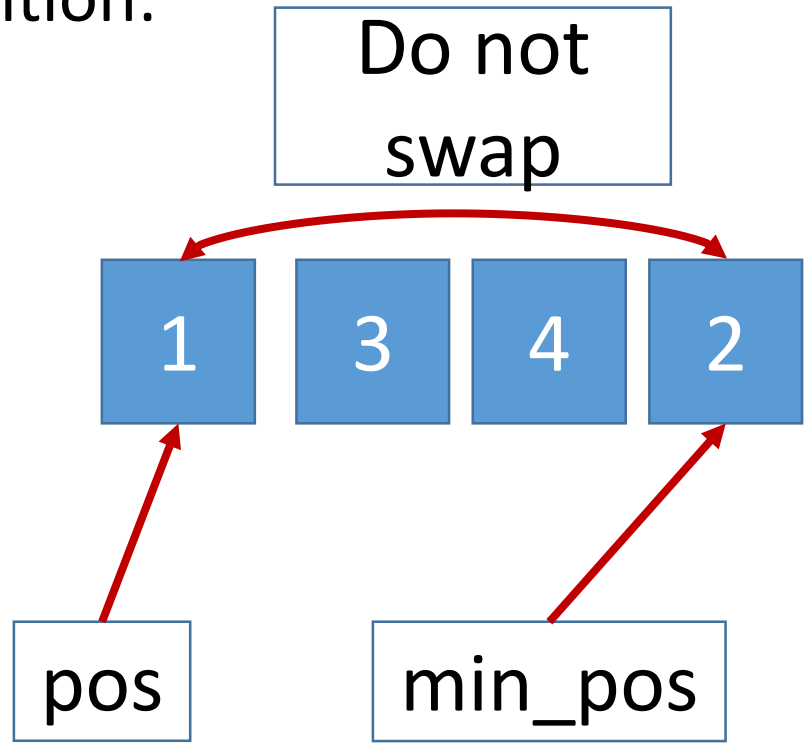
Intuition:



Sorting

- For each position in the array, pos
  - Let curr be the current value at pos
  - Find the position of the minimal element from pos to the end: min_val, min_pos
  - Swap the values of pos and min_pos

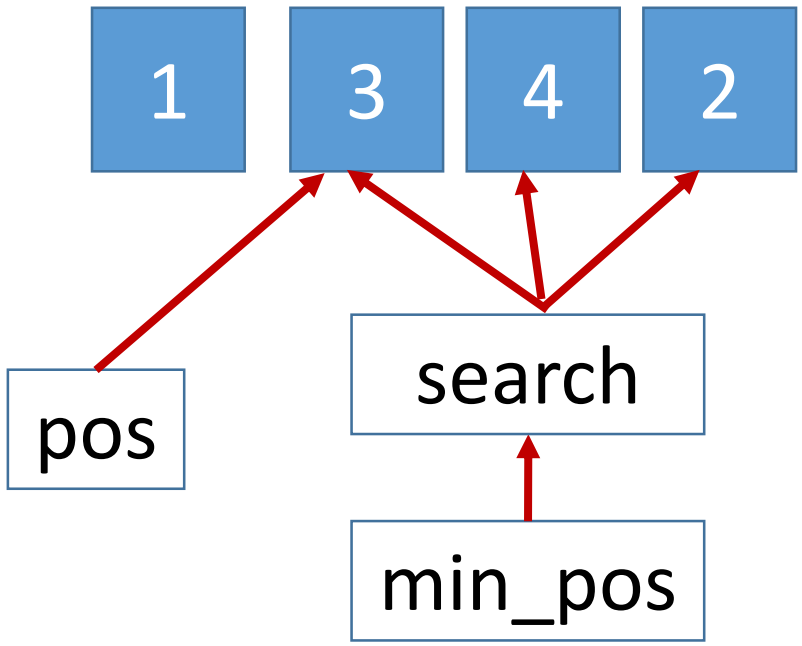
Intuition:



Sorting

- For each position in the array, pos
  - Let curr be the current value at pos
  - Find the position of the minimal element from pos to the end: min_val, min_pos
  - Swap the values of pos and min_pos

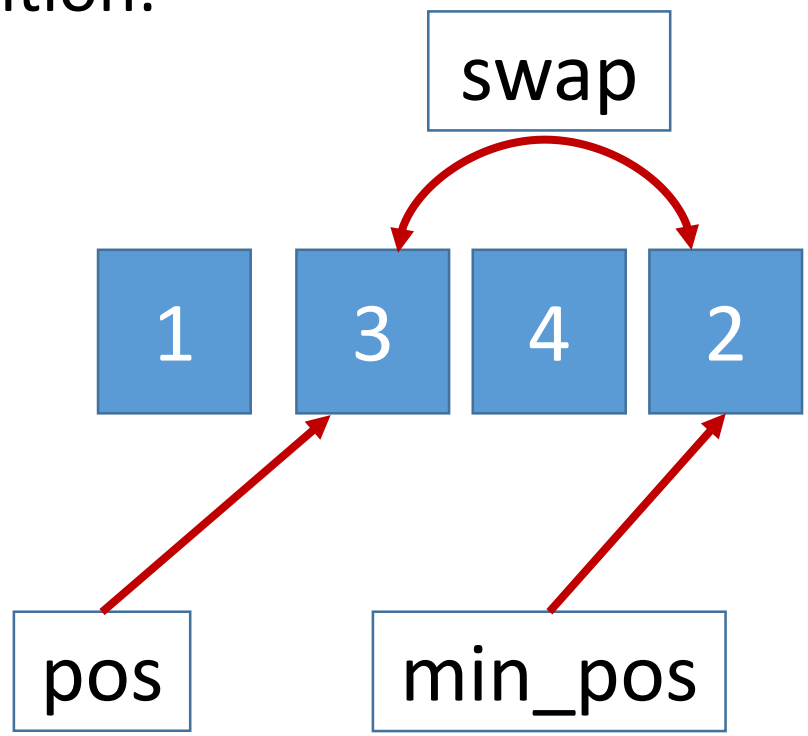
Intuition:



Sorting

- For each position in the array, pos
  - Let curr be the current value at pos
  - Find the position of the minimal element from pos to the end: min_val, min_pos
  - Swap the values of pos and min_pos

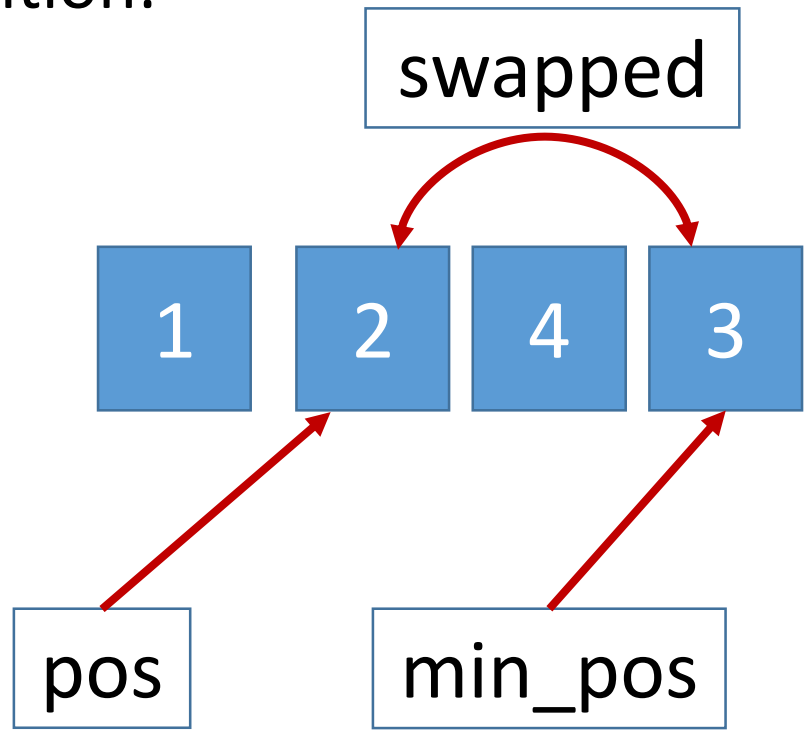
Intuition:



Sorting

- For each position in the array, pos
  - Let curr be the current value at pos
  - Find the position of the minimal element from pos to the end: min_val, min_pos
  - Swap the values of pos and min_pos

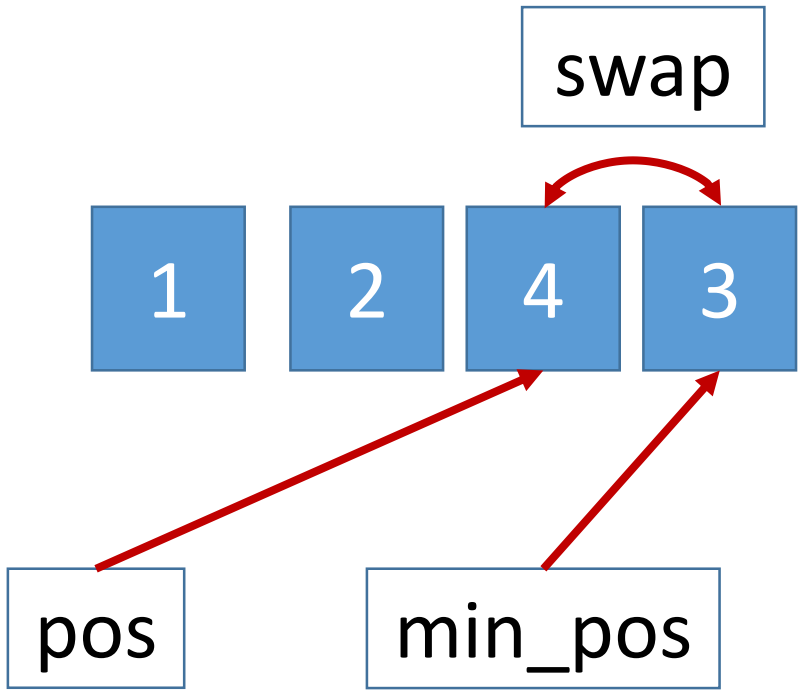
Intuition:



Sorting

- For each position in the array, pos
  - Let curr be the current value at pos
  - Find the position of the minimal element from pos to the end: min_val, min_pos
  - Swap the values of pos and min_pos

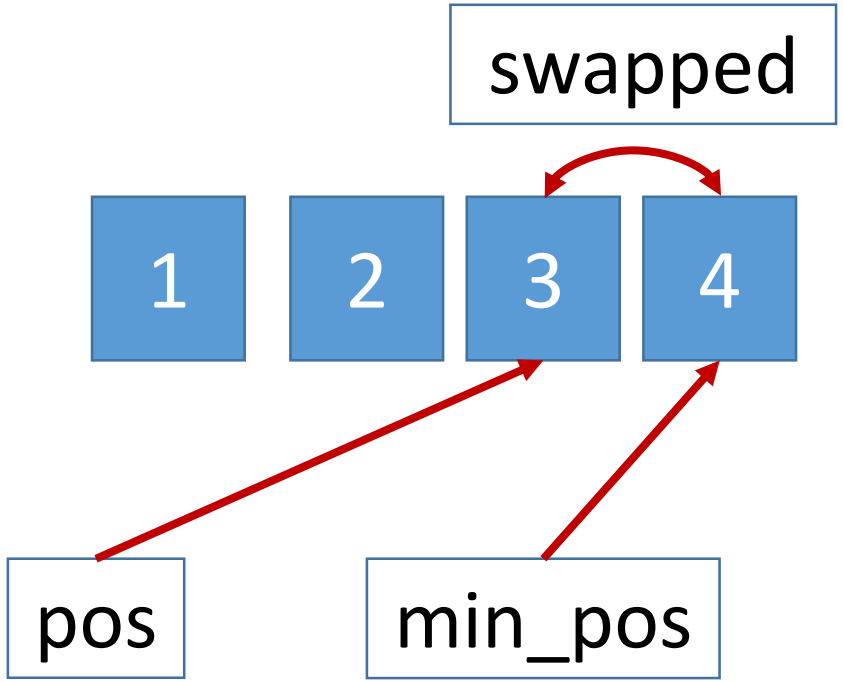
Intuition:



Sorting

- For each position in the array, pos
  - Let curr be the current value at pos
  - Find the position of the minimal element from pos to the end: min_val, min_pos
  - Swap the values of pos and min_pos

Intuition:



Dave's Solution in C

- Setup the array
- One for loop over the position that should next contain the minimum element
- Another loop to find the min
- An if to check about swapping
- Print at the end

```c
#include <stdio.h>

int main(){
        int array[4] = { 3, 1, 4, 2 };

        for( int pos=0; pos<3; pos++ ){
                int val_here = array[pos];
                int min_over_rest = 9999999;
                int min_pos = -1;

                for( int next_pos=pos+1; next_pos < 4; next_pos++ ){
                        if( array[next_pos] < min_over_rest ){
                                min_over_rest = array[next_pos];
                                min_pos = next_pos;
                        }
                }

                if( min_over_rest < val_here ){
                        array[ min_pos ] = val_here;
                        array[ pos ] = min_over_rest;
                }
        }

        for( int pos=0; pos<4; pos++ ){
                printf( "Array element %d was %d.\n", pos, array[pos] );

        }
}
```

Dave's Solution in C

- Setup the array
- One for loop over the position that should next contain the minimum element
- Another loop to find the min
- An if to check about swapping
- Print at the end

```c
#include <stdio.h>

int main(){
        int array[4] = { 3, 1, 4, 2 };

        for( int pos=0; pos<3; pos++ ){
                int val_here = array[pos];
                int min_over_rest = 9999999;
                int min_pos = -1;

                for( int next_pos=pos+1; next_pos < 4; next_pos++ ){
                        if( array[next_pos] < min_over_rest ){
                                min_over_rest = array[next_pos];
                                min_pos = next_pos;
                        }
                }

                if( min_over_rest < val_here ){
                        array[ min_pos ] = val_here;
                        array[ pos ] = min_over_rest;
                }
        }

        for( int pos=0; pos<4; pos++ ){
                printf( "Array element %d was %d.\n", pos, array[pos] );

        }
}
```

Dave's Solution in C

- Setup the array
- One for loop over the position that should next contain the minimum element
- Another loop to find the min
- An if to check about swapping
- Print at the end

```c
#include <stdio.h>

int main(){
        int array[4] = { 3, 1, 4, 2 };

        for( int pos=0; pos<3; pos++ ){
                int val_here = array[pos];
                int min_over_rest = 9999999;
                int min_pos = -1;

                for( int next_pos=pos+1; next_pos < 4; next_pos++ ){
                        if( array[next_pos] < min_over_rest ){
                                min_over_rest = array[next_pos];
                                min_pos = next_pos;
                        }
                }

                if( min_over_rest < val_here ){
                        array[ min_pos ] = val_here;
                        array[ pos ] = min_over_rest;
                }
        }

        for( int pos=0; pos<4; pos++ ){
                printf( "Array element %d was %d.\n", pos, array[pos] );

        }
}
```

Dave's Solution in C

- Setup the array
- One for loop over the position that should next contain the minimum element
- Another loop to find the min
- An if to check about swapping
- Print at the end

```c
#include <stdio.h>

int main(){
        int array[4] = { 3, 1, 4, 2 };

        for( int pos=0; pos<3; pos++ ){
                int val_here = array[pos];
                int min_over_rest = 9999999;
                int min_pos = -1;

                for( int next_pos=pos+1; next_pos < 4; next_pos++ ){
                        if( array[next_pos] < min_over_rest ){
                                min_over_rest = array[next_pos];
                                min_pos = next_pos;
                        }
                }

                if( min_over_rest < val_here ){
                        array[ min_pos ] = val_here;
                        array[ pos ] = min_over_rest;
                }
        }

        for( int pos=0; pos<4; pos++ ){
                printf( "Array element %d was %d.\n", pos, array[pos] );

        }
}
```

Dave's Solution in C

- Setup the array
- One for loop over the position that should next contain the minimum element
- Another loop to find the min
- An if to check about swapping
- Print at the end

```c
#include <stdio.h>

int main(){
        int array[4] = { 3, 1, 4, 2 };

        for( int pos=0; pos<3; pos++ ){
                int val_here = array[pos];
                int min_over_rest = 9999999;
                int min_pos = -1;

                for( int next_pos=pos+1; next_pos < 4; next_pos++ ){
                        if( array[next_pos] < min_over_rest ){
                                min_over_rest = array[next_pos];
                                min_pos = next_pos;
                        }
                }

                if( min_over_rest < val_here ){
                        array[ min_pos ] = val_here;
                        array[ pos ] = min_over_rest;
                }
        }

        for( int pos=0; pos<4; pos++ ){
                printf( "Array element %d was %d.\n", pos, array[pos] );

        }
}
```

A bit more “programmatic” version

- In 206, we hate to hard-code
- I wrote the array size, its contents, and a number of constant sizes throughout my simple solution
- What about taking any number of numbers from the user as command-line arguments and sorting that list?

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

int main( int argc, char* argv[] ){

        int array[argc-1];

        for( int pos=1; pos<argc; pos++ )
                array[pos-1] = atoi( argv[pos] );

        for( int pos=0; pos<argc-1; pos++ ){
                int curr = array[pos];
                int min_pos = -1;
                int min_val = INT_MAX;

                for( int other_pos=pos; other_pos<argc-1; other_pos++ ){
                        if( array[other_pos] < min_val ){
                                min_pos = other_pos;
                                min_val = array[other_pos];
                        }
                }

                array[min_pos] = curr;
                array[pos] = min_val;
        }


        for( int pos=0; pos<argc-1; pos++ )
                printf( "Array element %d is %d.\n", pos, array[pos] );

        return 0;

}
```

Remember, argc gives you the number of arguments in variable form

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

int main( int argc, char* argv[] ){

        int array[argc-1];

        for( int pos=1; pos<argc; pos++ )
                array[pos-1] = atoi( argv[pos] );

        for( int pos=0; pos<argc-1; pos++ ){
                int curr = array[pos];
                int min_pos = -1;
                int min_val = INT_MAX;

                for( int other_pos=pos; other_pos<argc-1; other_pos++ ){
                        if( array[other_pos] < min_val ){
                                min_pos = other_pos;
                                min_val = array[other_pos];
                        }
                }

                array[min_pos] = curr;
                array[pos] = min_val;
        }


        for( int pos=0; pos<argc-1; pos++ )
                printf( "Array element %d is %d.\n", pos, array[pos] );

        return 0;

}
```

atoi() converts a string into the number it represents (-1 if there was a failure)

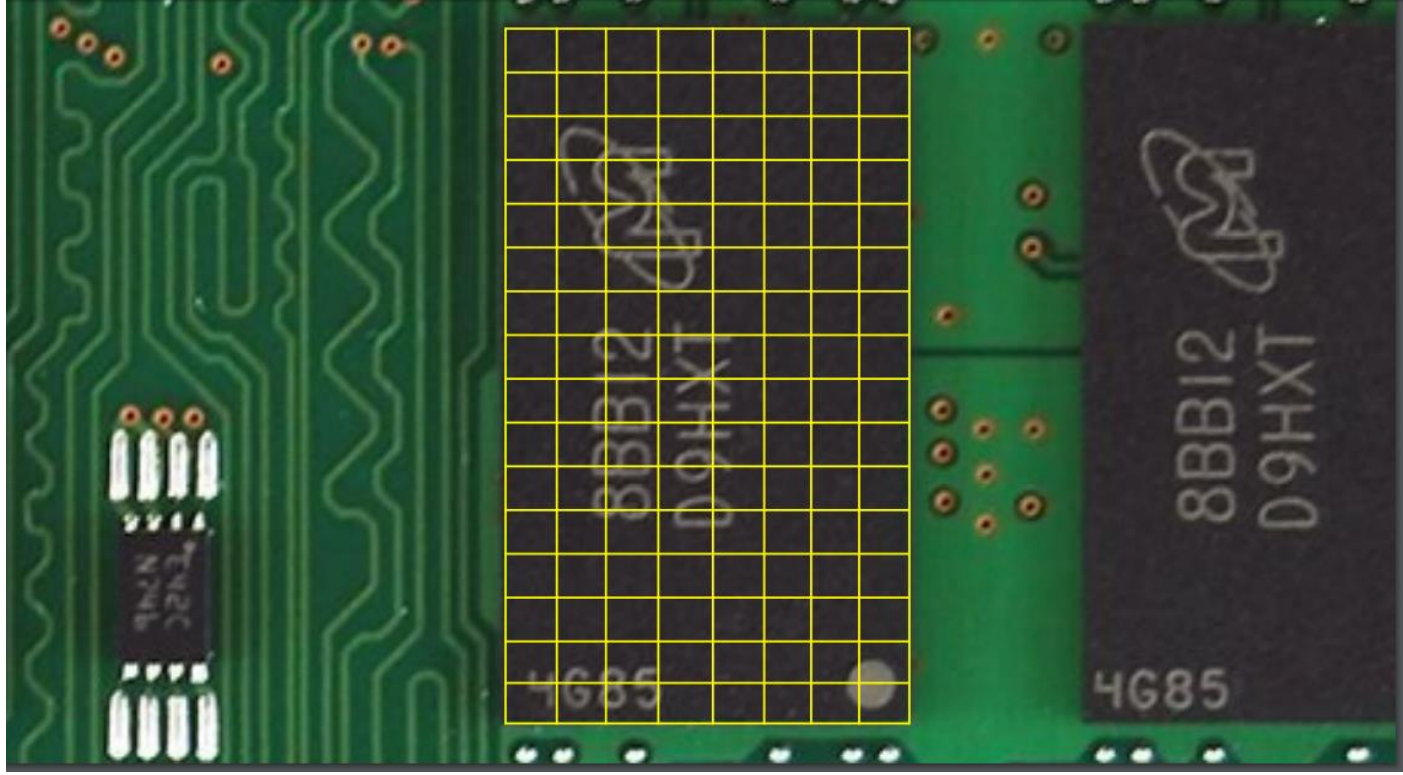
```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int main( int argc, char* argv[] ){

        int array[argc-1];

        for( int pos=1; pos<argc; pos++ )
                array[pos-1] = atoi( argv[pos] );

        for( int pos=0; pos<argc-1; pos++ ){
                int curr = array[pos];
                int min_pos = -1;
                int min_val = INT_MAX;

                for( int other_pos=pos; other_pos<argc-1; other_pos++ ){
                        if( array[other_pos] < min_val ){
                                min_pos = other_pos;
                                min_val = array[other_pos];
                        }
                }

                array[min_pos] = curr;
                array[pos] = min_val;
        }


        for( int pos=0; pos<argc-1; pos++ )
                printf( "Array element %d is %d.\n", pos, array[pos] );

        return 0;

}
```

INT_MAX is better than using 999999. It's guaranteed to hold the largest int value.

Focus Today: Data, data, data

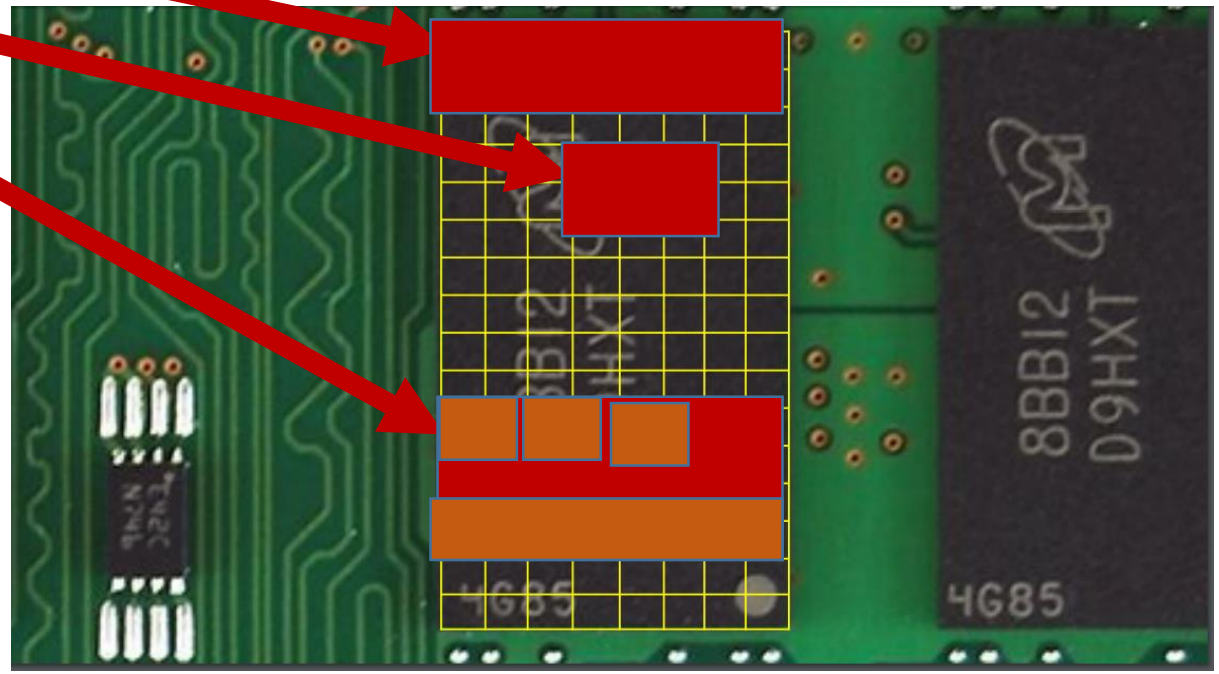
All data is binary

Organized using addresses



What’s in memory: we will understand all of this by the end of the class

- The kernel process
- The shell process
- Your program’s process
  - The program’s machine code
  - Program variables
  - Required libraries
  - Book-keeping info



- For today: we start with the data in a C program

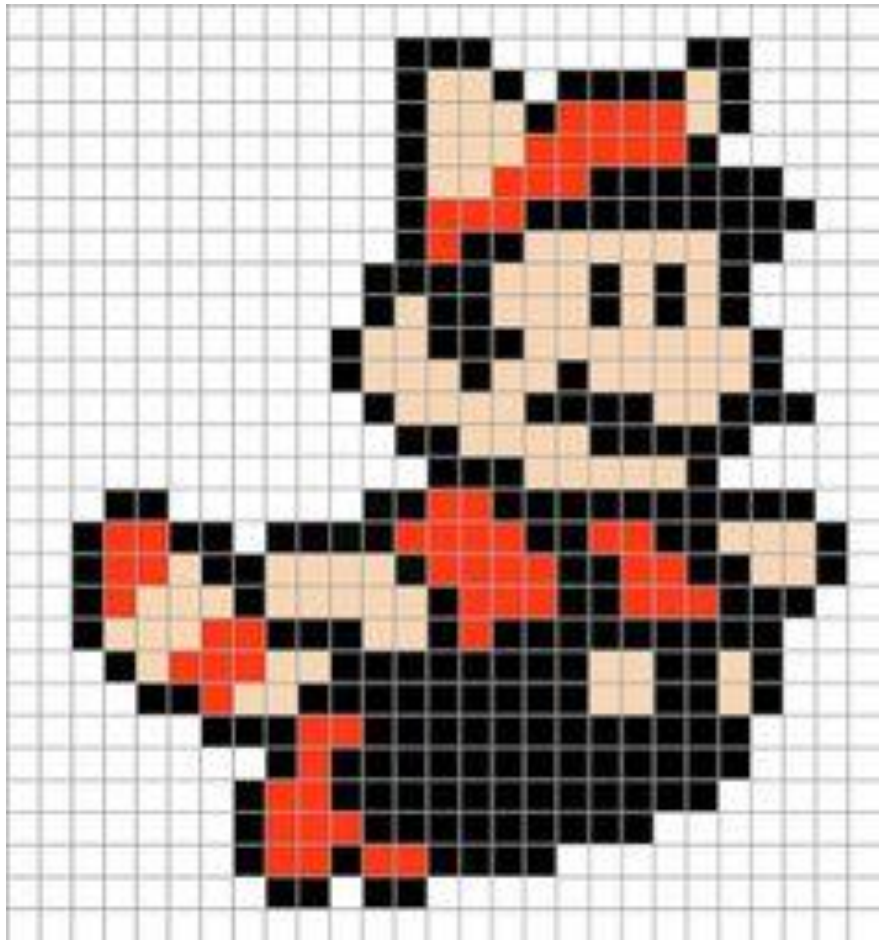
How much space does a variable take?

Description	Type	Bits	Range
Integer	short int long	16 32 64	+/-32 thousand +/-2.1 billion +- 9.2 x 10^{18}
Floating point	float double long double	32 64 128	+/- 10^{38} +/- 10^{308} +/- 10^{4932}
Character	char unsigned char	8	-127 to 128 0 to 255
Pointer	char* int* (etc)	64	0 to 1.8 x 10^{19}

Choosing the right type

- Match between the meaning of the underlying data and the available tools and data types from C

- Example: 16-bit color:
  - Red takes 4 bits
  - Green takes 4 bits
  - Blue takes 4 bits
  - Transparency takes 4 bits
  - Choices: 2 chars, 1 short, 4 chars, 1 int, etc

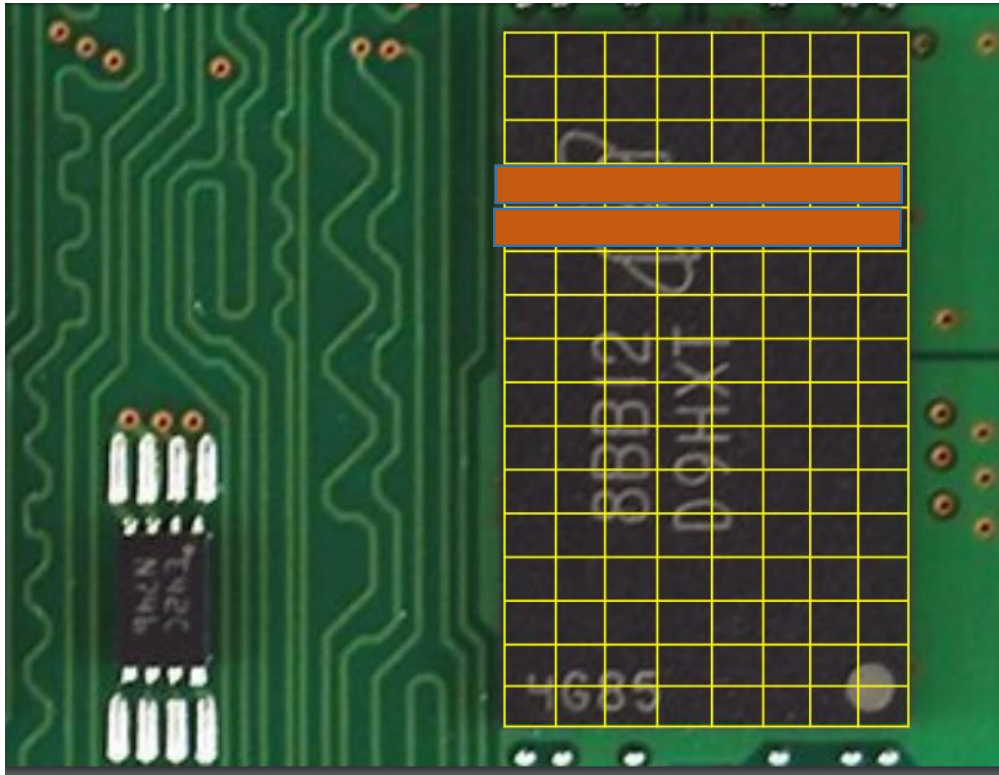


What if we try to store something larger?

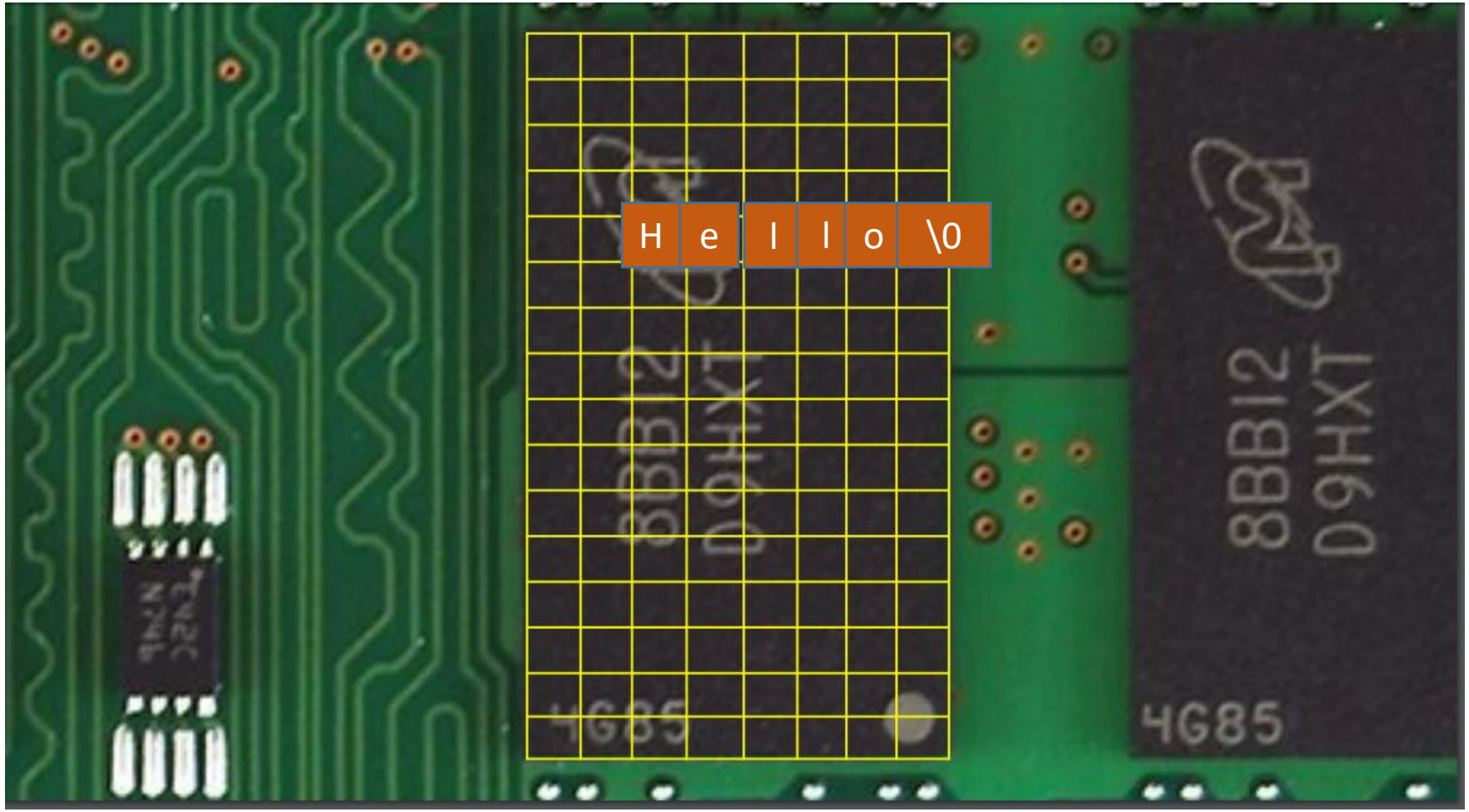
```
int i=0;
while(1){
   printf( “i is %d.\n”, i );
   i++;
}
```

Arrays and Memory

- Arrays take multiple “slots”, each of the underlying type
- They are guaranteed to be stored in order
- The number as you specify on array creation and does not change



Strings in memory



Memory is binary

- Char type is really an 8-bit integer
- The mapping to text characters is defined by another standard!

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

C Strings

- Arrays where each element is a character stored using the ASCII table
- Must be “null terminated” with the special ‘\0’ NULL value, with integer representation 0
- This allows things like printf to output just the data you want
  - Let’s think about that

Exercises

- Write a C program to reverse the characters in its first argument and output to terminal
- Re-write our sort program to sort text words instead of numbers
- Write a program that tells you the number of characters in a word, the number of words on a line, or the number of lines in a file (requires reading ahead)

- Read the first 3 chapters of K&R text.