CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 10.2: Struct example, binary file I/O

Toronto Metropolitan University

Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

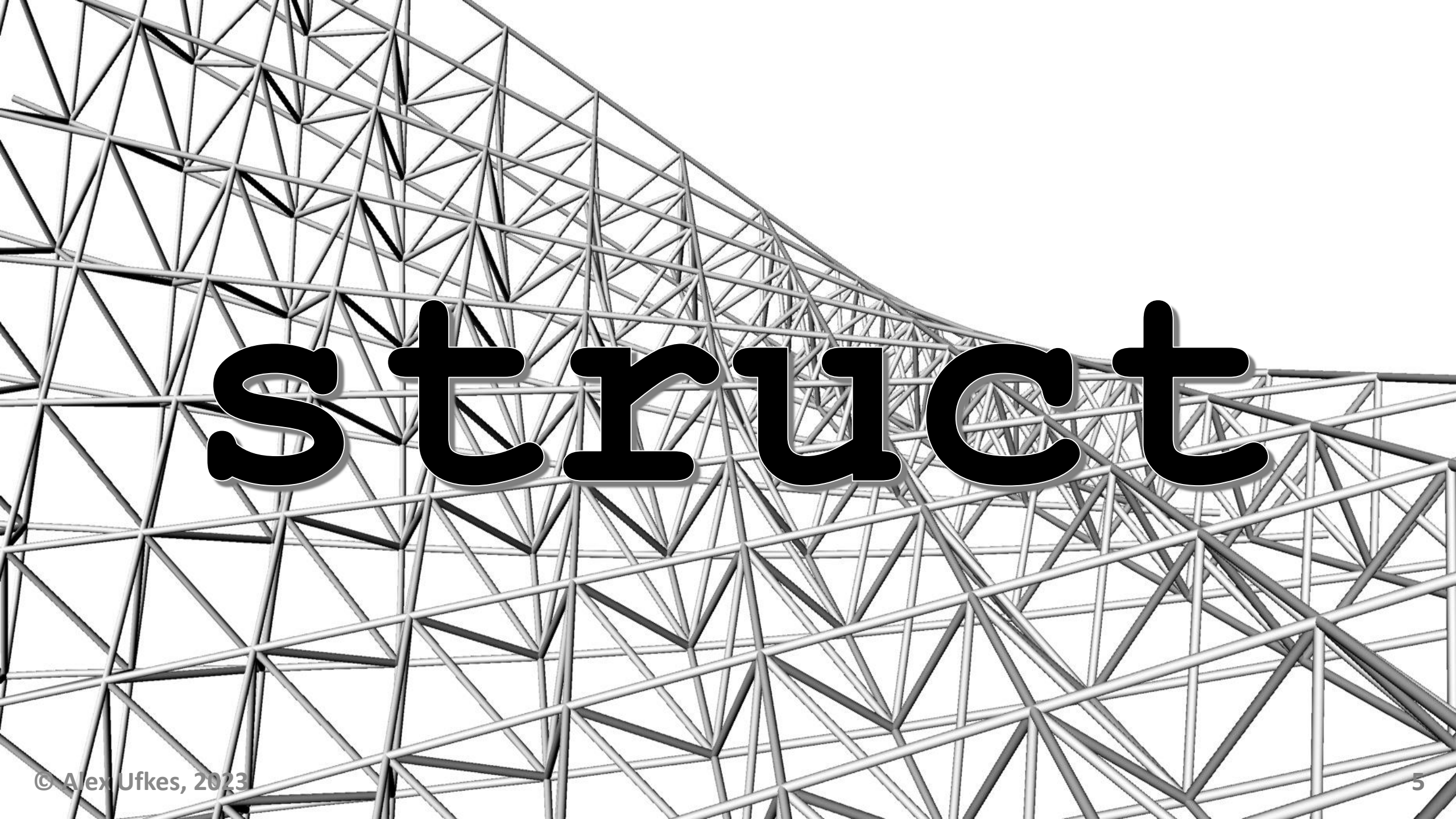
The copyright to this original work is held by Alex Ufkes. Students registered in course CPS 188 can use this material for the purposes of this course but no other use is permitted, and there can be no sale or transfer or use of the work for any other purpose without explicit permission of Alex Ufkes.

Notice!

Final exam info posted on D2L.

Check the announcement!

Previously…



Structs

A collection of variables, types can vary

```
struct ball
{
    char style[10];
    double radius;
    double weight;
};
```

```
struct ball
{
    char style[10];
    double radius;
    double weight;
};

int main()
{
    struct ball b1, b2, b3, b[50];

}
```

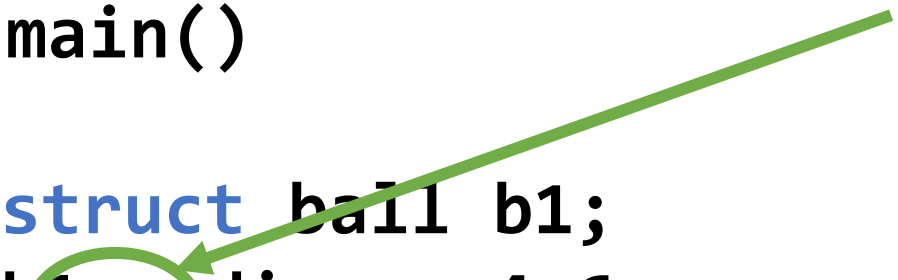
`struct` variable declaration

Or as an array

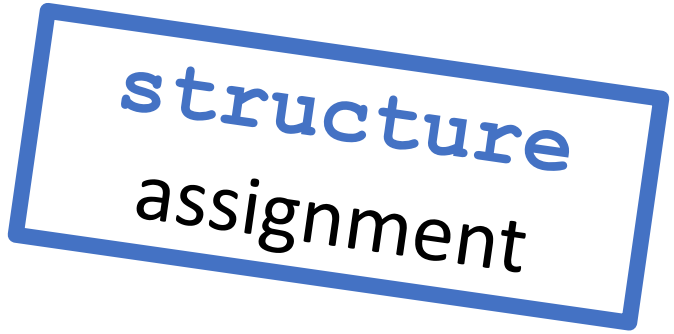
Declare individually

Accessing `struct` members

```
struct ball
{
    char style[10];
    double radius;
    double weight;
};

int main()
{
    struct ball b1;
    b1.radius = 4.6;
    b1.weight = 22.1;
    strcpy(b1.style, "Wilson");
}
```



Use the dot operator to access a struct's individual variables.




```
struct1 = struct2;
```

Assigns one struct to another

Assuming the same structure type

All member variable values are copied!

typedef

Create an Alias

```
#include <stdio.h>

typedef int integer;

int main()
{
    integer x, y, z;
    x = 1;
    y = 2;
    z = x + y;
}
```

Create "integer" alias for type int

`integer` can now be used as a type! It's the same as `int`, except for the name.

Common for Structures

```
typedef struct ball
{
    char style[10];
    double radius;
    double weight;
} ball;
```

Now, instead of declaring:

```
struct ball b1, b2, b3;
```

We can simply declare:

```
ball b1, b2, b3;
```

Common for Structures

Alias can be different from struct name:

```
typedef struct ball
{
    char style[10];
    double radius;
    double weight;
} basketball;
```

Declaration:

```
basketball b1, b2, b3;
```

Can still declare:

```
struct ball a, b, c;
```

```
typedef struct ball
{
    char style[10];
    double radius;
    double weight;
} basketball;

int main()
{
     basketball b1 = {“Wilson”, 3.4, 6.8};
     return 0;
}
```

Declare and Initialize

Order must match structure definition

Structures & Functions

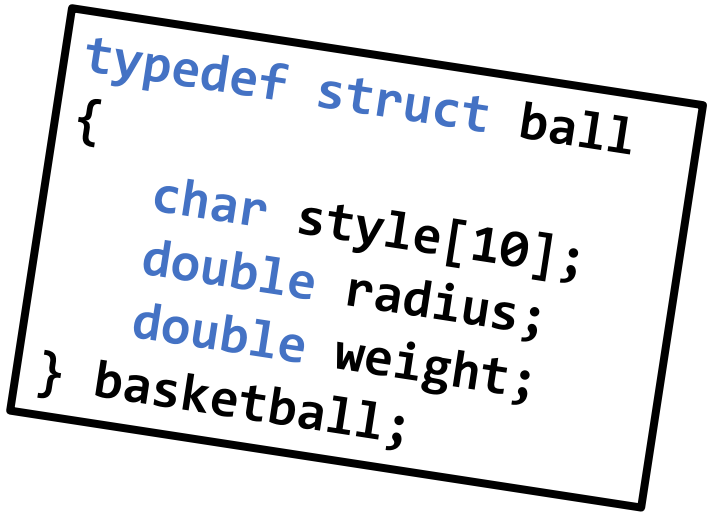
Structures passed as input to a function are **COPIED** into the corresponding function parameter. All member variables are copied as well.

This is **DIFFERENT** from arrays, which are **NOT** copied. Only the base address of the array is copied, not the elements.

Write a user-defined function that does the following:

Takes in zero arguments, and returns a `basketball` struct.
The function will ask the user to enter values for each of `basketball`'s member variables.

```
basketball readElements(void)
{
    basketball b;
    printf("Enter the style: ");
    scanf("%s", b.style);
    printf("Enter the radius: ");
    scanf("%lf", &b.radius);
    printf("Enter the weight: ");
    scanf("%lf", &b.weight);
    return b;
}
```



```
basketball readElements(void)
{
        basketball b;
        printf("Enter the style: ");
        scanf("%s", b.style);
        printf("Enter the radius: ");
        scanf("%lf", &b.radius);
        printf("Enter the weight: ");
        scanf("%lf", &b.weight);
        return b;
}
```

```
void printElements(basketball b)
{
    printf("Style:  %s\n", b.style);
    printf("Radius: %lf\n", b.radius);
    printf("Weight: %lf\n", b.weight);
}
```

It is possible to return an entire `struct`. ***Unlike*** arrays, ALL values are copied.

```
int main()
{
        basketball b1 = readElements();
        printElements(b1);
        return 0;
}
```

Pass a pointer:

```
typedef struct foo {
    int a, b;
} foo;
```

```
void swap(foo input)
{
    int tmp = input.a;
    input.a = input.b;
    input.b = tmp;
}
```

```
void swap(foo *input)
{
    int tmp = input->a;
    input->a = input->b;
    input->b = tmp;
}
```

If `input` is a ***pointer*** to a struct, we use the **arrow operator (->)** instead of the dot operator.

```
void swap(foo *input)
{
    int tmp = input->a;
    input->a = input->b;
    input->b = tmp;
}

int main()
{
     foo f1 = {1, 2};
     printf("Before swap: %d %d\n", f1.a, f1.b);
     swap(&f1);
     printf("After swap:  %d %d", f1.a, f1.b);
     return 0;
}
```

What is the output?

```
Before swap: 1 2
After swap:  2 1
```

We can also dereference!

```
void printElements(basketball *b)
{
    printf("Style:  %s\n",  b->style);
    printf("Radius: %lf\n", b->radius);
    printf("Weight: %lf\n", b->weight);
}
```

```
typedef struct ball
{
    char style[10];
    double radius;
    double weight;
} basketball;
```

Is the same as:

```
void printElements(basketball *b)
{
    printf("Style:  %s\n",  (*b).style);
    printf("Radius: %lf\n", (*b).radius);
    printf("Weight: %lf\n", (*b).weight);
}
```

Must use parentheses (*b) otherwise the dot operator is applied first

Example: Complex Number

$$\sqrt{-1}$$

Example: Complex Number

```
typedef struct complex
{
    double real;
    double imag;
} complex;
```

Complex Number: Scanning

```
complex scan_complex ()
{
    complex c;

    printf("Enter real component: ");
    scanf("%lf", &c.real);
    printf("Enter imaginary component: ");
    scanf("%lf", &c.imag);

    return (c);
}
```

Complex Number: Printing

```
void print_complex (complex c)
{
    printf("%.2lf", c.real);
    if (c.imag >= 0.0)
        printf("+");
    printf("%.2lfi", c.imag);
}
```

Complex Number: Adding

Return a new complex number, whose components
are the sum of the operands:

```
complex add_complex (complex op1, complex op2)
{
    complex sum;
    sum.real = op1.real + op2.real;
    sum.imag = op1.imag + op2.imag;
    return sum;
}
```

Complex Number: Subtracting

Return a new complex number, whose components are the difference of the operands:

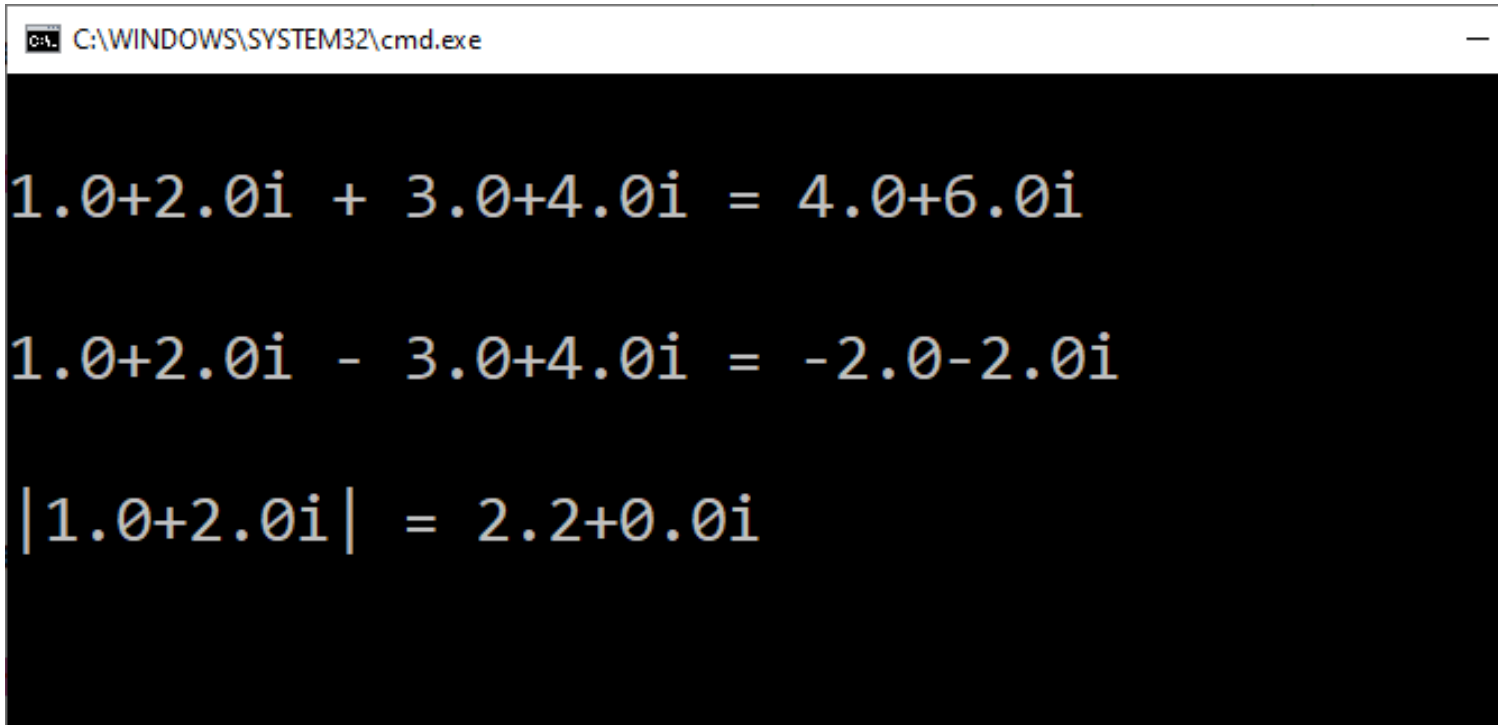
```
complex sub_complex (complex op1, complex op2)
{
    complex diff;
    diff.real = op1.real - op2.real;
    diff.imag = op1.imag - op2.imag;
    return diff;
}
```

Complex Number: Absolute Value

Hypotenuse between real and imaginary magnitudes:

```
complex abs_complex (complex c)
{
    complex res;
    res.real = sqrt(c.real*c.real + c.imag*c.imag);
    res.imag = 0;
    return res;
}
```

```
int main (void)
{
    complex c1 = {1, 2};
    complex c2 = {3, 4};

    /* Forms and displays the sum */
    printf("\n");
    print_complex(c1);
    printf(" + ");
    print_complex(c2);
    printf(" = ");
    print_complex(add_complex(c1, c2));

    /* Forms and displays the difference */
    printf("\n\n");
    print_complex(c1);
    printf(" - ");
    print_complex(c2);
    printf(" = ");
    print_complex(sub_complex(c1, c2));

    /* Forms and displays the absolute value of the first number */
    printf("\n\n|");
    print_complex(c1);
    printf("| = ");
    print_complex(abs_complex(c1));
    printf("\n");

    return (0);
}
```



Binary File IO

Binary VS ASCII

- Up to now, we've dealt exclusively with plain text files.
- Integers are read in the form of digit strings and converted internally.
- A text file containing **123456789** doesn't contain the *integer* 123456789.
- It contains the *digit characters* **'1'**, **'2'**, **'3'**, and so on.

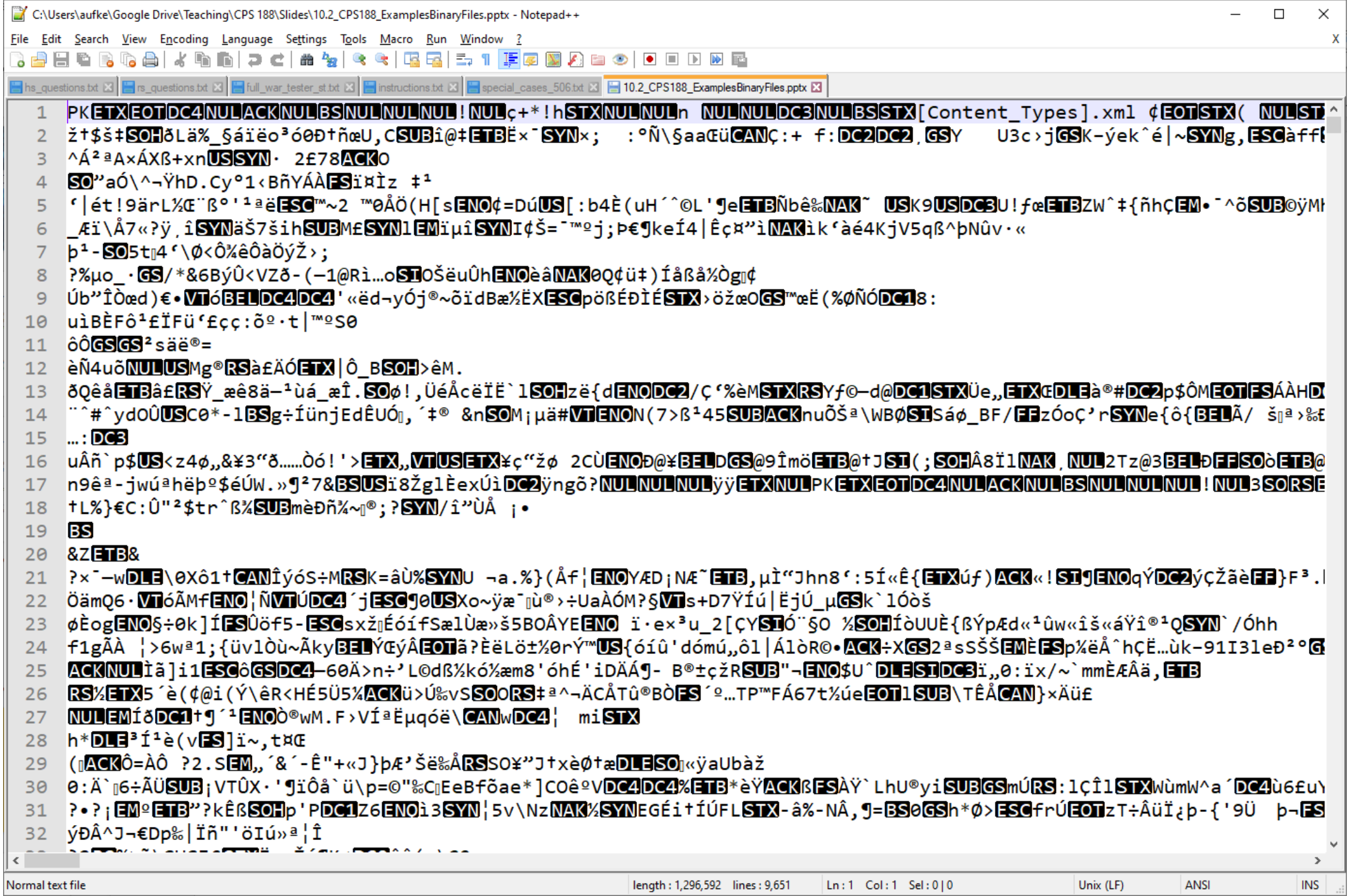
This is inefficient!

- Nine characters require 9 bytes of storage.
- 123456789 can also fit in an `int`, requiring only *four* bytes.
- Why not store this number as a binary, two's complement integer instead?

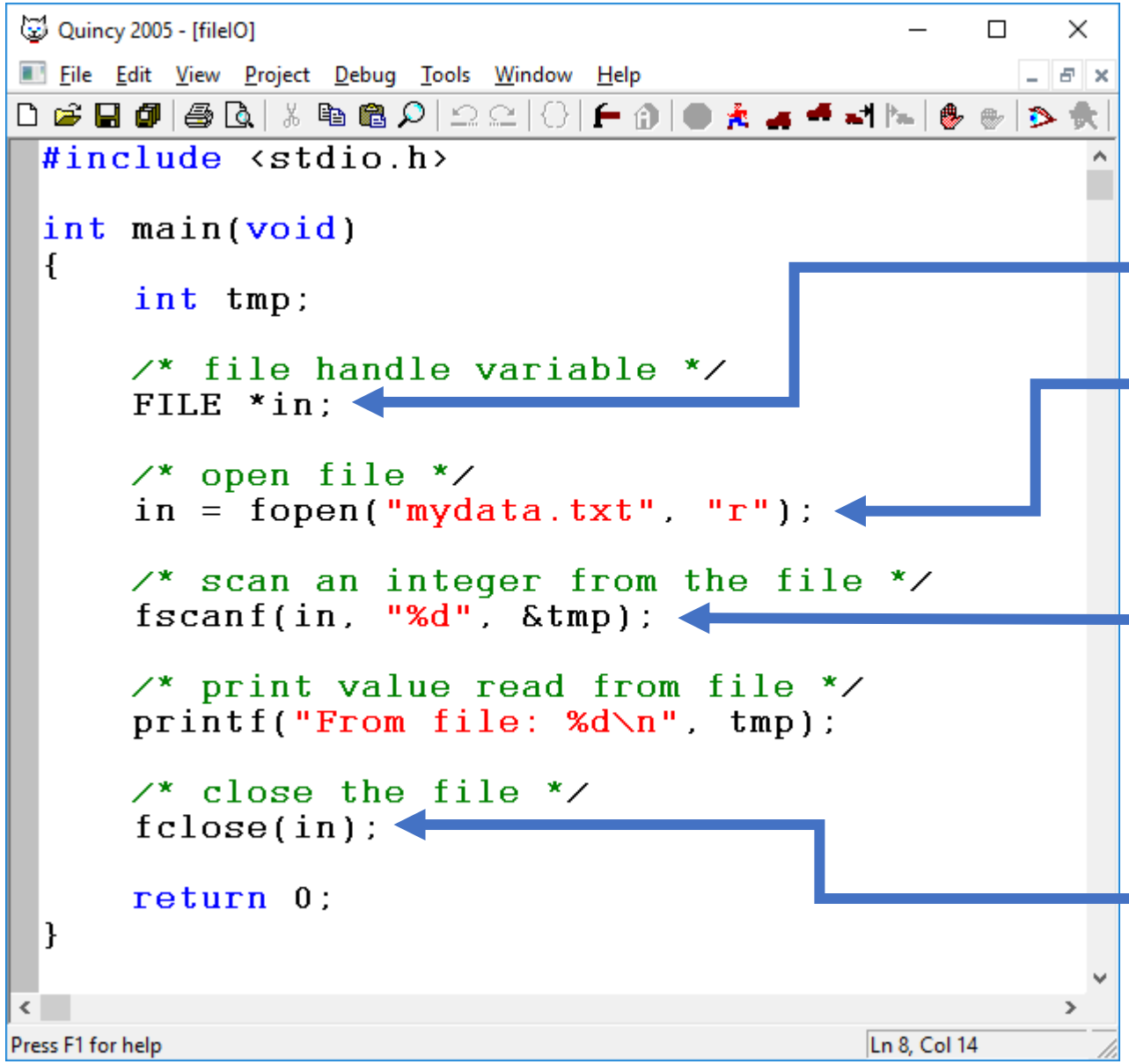
Binary Files

This will come with a downside!

- Plain text files are nice, because they are human-readable.
- A binary file is not. Its contents do not convert to readable ASCII characters.
- However, we can store information much more efficiently.
- This is why plain text files are actually very uncommon in commercial applications.
- Try opening a PDF, docx, excel spreadsheet, etc. as plain text.
- You get gibberish, because these file types use a different binary encoding.


C:\Users\aufke\Google Drive\Teaching\CPS 188\Slides\10.2_CPS188_ExamplesBinaryFiles.pptx - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Window ?
hs_questions.txt
rs_questions.txt
full_war_tester_st.txt
instructions.txt
special_cases_506.txt
10.2_CPS188_ExamplesBinaryFiles.pptx
PK ETX EOT DC4 NUL ACK NUL BS NUL NUL NUL ! NUL ç+*!h STX NUL NUL n NUL NUL DC3 NUL BS STX [Content_Types].xml ¢ EOT STX (NUL ST
ž†$š‡ SOH ðLä%_§áïëo³óØÐ†ñœU,C SUB î@‡ ETB Ë× ¯ SYN ×; :°Ñ\§aaŒü CAN Ç:+ f: DC2 DC2 , GS Y U3c›j GS K-ýek^é|~ SYN g, ESC àff
^Á²ªA×ÁXß+xn US SYN · 2£78 ACK O
SO ”aÓ\^¬ŸhD.Cy°1‹BñYÁÀ FS ï¤Ìz ‡¹
‘|ét!9ärL½Œ¨ß°'¹ªë ESC ™~2 ™0ÅÖ(H[s ENO ¢=Dú US [:b4È(uH´^©L'¶e ETB Ñbê‰ NAK ˜ US K9 US DC3 U!ƒœ ETB ZW^‡{ñhÇ EM •¯^õ SUB ©ÿM
_Æï\Å7«?ÿ¸î SYN äŠ7ših SUB M£ SYN l EM ïµî SYN I¢Š=¯™ºj;Þ€¶keÍ4|Êç¤”ì NAK ìk‘àé4KjV5qß^þNûv·«
þ¹- SO 5t 4‘\Ø<Ô¾êÔàÖýŽ›;
?%µo_· GS /*&6BýÛ<VZð-(–1@Rì…o SI OŠëuÛh ENO èâ NAK 0Q¢ü‡)Íåßå½Òg ¢
Úb”ÎÒœd)€• VT ó BEL DC4 DC4 '«ëd¬yÓj®~õïdBæ½ËX ESC pößÉÐÌÉ STX ›öžœO GS ™œË(%ØÑÓ DC1 8:
uìBÈFô¹£ÏFü‘£çç:õº·t|™ºS0
ôÔ GS GS ²säë®=
èÑ4uõ NUL US Mg® RS à£ÄÓ ETX |Ô_B SOH >êM.
ðQêå ETB â£ RS Ÿ_æê8ä–¹ùá_æÎ.SOø!,ÜéÅcëÏË`l SOH zë{d ENO DC2 /Ç‘%èM STX RS Yƒ©–d@ DC1 STX Üe„ ETX Œ DLE à®# DC2 p$ÔM EOT FS ÁÀH D
¨^#^ydOÛ US C0*-l BS g÷ÍünjEdÊUÓ ,´‡® &n SO M¡µä# VT ENO N(7>ß¹45 SUB ACK nuÕŠª\WBØ SI Sáø_BF/ FF zÓoÇ’r SYN e{ô{ BEL Ã/ š ª›‰£
…: DC3
uÂñ`p$ US <z4ø„&¥3“ð……Òó!'> ETX „ VT US ETX ¥ç“žø 2CÙ ENO Ð@¥ BEL D GS @9Îmö ETB @†J SI (; SOH Â8Ï1 NAK , NUL 2Tz@3 BEL Ð FF SO ò ETB @
n9êª-jwúªhëþº$éÚW.»¶²7& BS US ï8Žgl ÈexÚì DC2 ÿngõ? NUL NUL NUL ÿÿ ETX NUL PK ETX EOT DC4 NUL ACK NUL BS NUL NUL NUL ! NUL 3 SO RS E
†L%}€C:Û"²$tr^ß¾ SUB mèÐñ¾~ ®;? SYN /î”ÙÅ ¡•
BS
&Z ETB &
?×¯–w DLE \0Xô1† CAN ÎýóS÷M RS K=âÙ% SYN U ¬a.%}(Åf¦ ENO YÆD¡NÆ˜ ETB ,µÌ“Jhn8‘:5Í«Ê{ ETX úƒ) ACK «! SI ¶ ENO qÝ DC2 ýÇŽãè FF }F³.
ÖämQ6· VT óÃMf ENO ¦Ñ VT Ú DC4 ´j ESC ¶0 US Xo~ÿæ¯ ù®›÷UaÀÓM?§ VT s+D7ŸÍú|ËjÚ_µ GS k`lÓòš
øÈog ENO §÷0k]Í FS Ûöf5- ESC sxž ÉóífSælÙæ»š5BOÂYE ENO ï·ex³u_2[ÇY SI Ó¨§O ½ SOH ÍòUUÈ{ßÝpÆd«¹ûw«îš«áŸî®¹Q SYN `/Óhh
f1gÃÀ ¦>6wª1;{üvlÒù~Ãky BEL ÝŒýÂ EOT ã?ÈëLö±½0rÝ™ US {óíû'dómú„ôl|ÁlòR©• ACK ÷X GS 2ªsSŠŠ EM È FS p¼ëÅ^hÇË…ùk-91I3leÐ²° G
ACK NUL Ìã]i1 ESC ô GS DC4 –60Ä>n÷’L©dß½kó½æm8'óhÉ'iDÄÁ¶- B®±çžR SUB "¬ ENO $U^ DLE SI DC3 ï„0:ïx/~`mmÈÆÂä, ETB
RS ½ ETX 5´è(¢@i(Ý\êR<HÉ5Ü5¾ ACK ü>Ú‰vS SO O RS ‡ª^¬ÄCÅTû®BÒ FS ´º…TP™FÁ67t½úe EOT l SUB \TÊÅ CAN }×Äü£
NUL EM Íð DC1 †¶´¹ ENO Ò®wM.F›VÍªËµqóë\ CAN w DC4 ¦ mi STX
h* DLE ³Í¹è(v FS]ï~,t¤Œ
(ACK Ô=ÀÔ ?2.S EM „´&´-Ê"+«J}þÆ’Šë‰Å RS SO¥”J†xèØ†æ DLE SO «ÿaUbàž
0:Ä` 6÷ÃÜ SUB ¡VTÛX·'¶ïÔå`ü\p=©"‰C EeBfõae*]COêºV DC4 DC4 %ETB*èŸ ACK ß FS ÀŸ`LhU®yi SUB GS mÚ RS :lÇÎl STX WùmW^a´ DC4 ù6£u
?•?¡ EM º ETB ”?kÊß SOH p'P DC1 Z6 ENO ì3 SYN ¦5v\Nz NAK ½ SYN EGÉi†ÍÚFL STX -â%-NÂ,¶=BS0 GS h*Ø>ESC frÚ EOT zT÷ÂüÏ¿þ-{'9Ü þ¬ FS
ýÐÂ^J¬€Dp‰|Ïñ"'öIú»ª¦Î
Normal text file
length : 1,296,592 lines : 9,651
Ln : 1 Col : 1 Sel : 0 | 0
Unix (LF)
ANSI
INS

fscanf



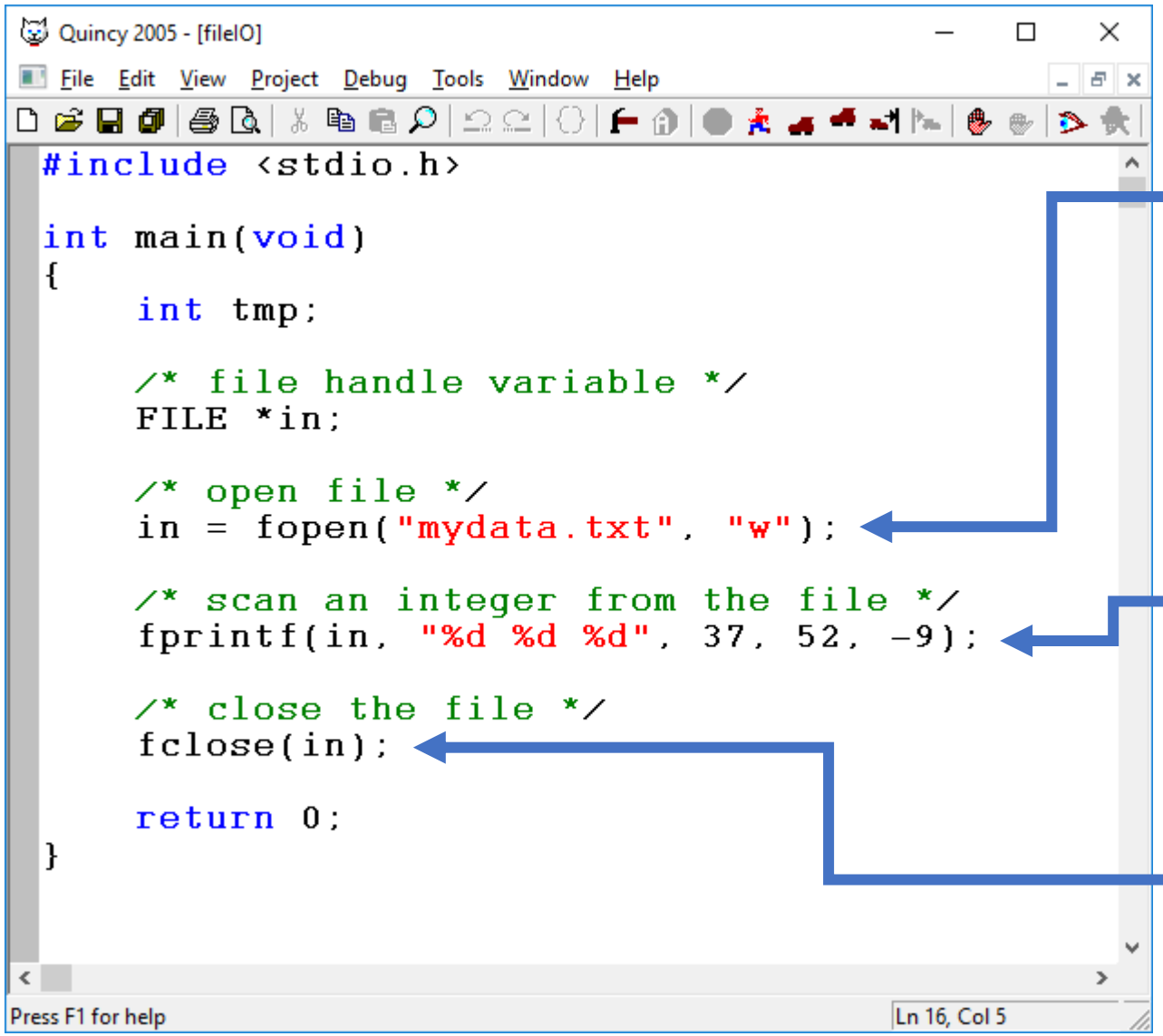
New variable type: FILE *

fopen to open the file. First argument is the filename, second is the mode. **"r"** means ***read***.

fscanf - Like scanf but takes an argument before the string: the file handle variable.

fclose – File should be closed once we're done with it.

fprintf



Must first open the file:

- This time we're ***writing***. "w" means write.
- ***If the file already exists, it will be overwritten!***

fprintf - Like printf but takes an argument before the string: the file handle variable.

fclose – File should be closed once we're done with it.

Read Binary, Write Binary

We still use **fopen**, but this time the 2nd argument is different:

```
fopen("data.bin", "wb");
```

"wb" for write binary

```
fopen("data.bin", "rb");
```

"rb" for read binary

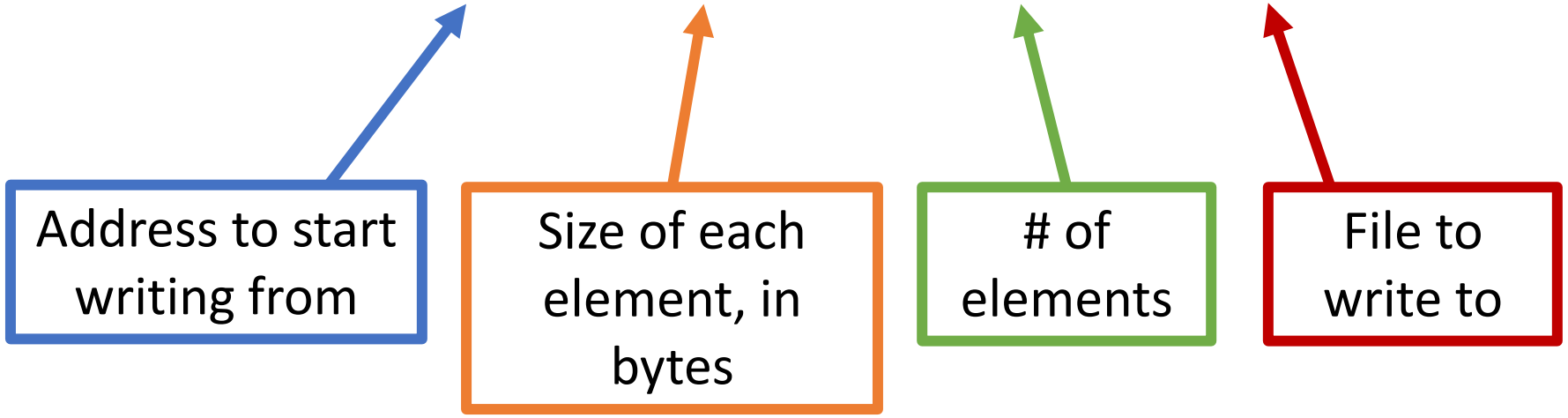
File extension?

- For binary files, **.dat** is common, as is **.bin**
- Extension doesn't matter! We can read the bits in a file however we want.

Read Binary, Write Binary

We no longer use **`fprintf`** or **`fscanf`**:

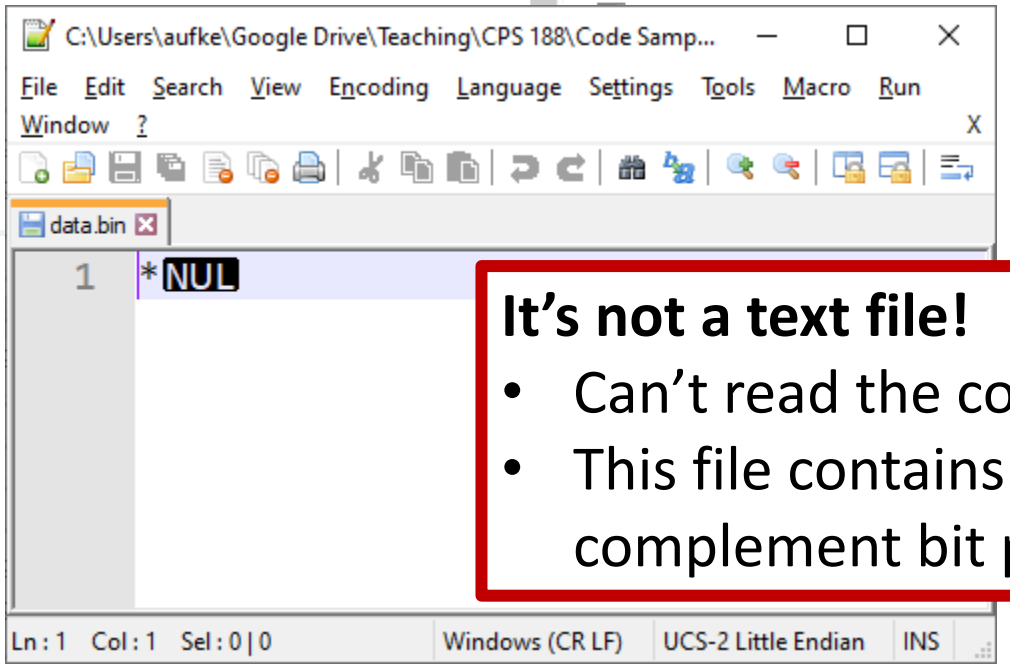
```
FILE* bin_out;
bin_out = fopen("data.bin", "wb");
int x = 42
fwrite(&x, sizeof(x), 1, bin_out);
```



binaryIO.c

```c
#include <stdio.h>

int main (void)
{
    FILE* bin_out = fopen("data.bin", "wb");

    int x = 42;

    fwrite(&x, sizeof(x), 1, bin_out);

    fclose(bin_out);

    return (0);
}

```

Name	Date modified	Type	Size
binaryIO.c	3/24/2023 8:23 AM	C File	1 KB
binaryIO.exe	3/24/2023 8:24 AM	Application	120 KB
binsearch.exe	2/22/2023 10:44 AM	Application	249 KB
circ_calc.c	1/26/2023 11:03 AM	C File	1 KB
circ_calc.exe	1/26/2023 11:03 AM	Application	405 KB
close_enough.c	2/2/2023 2:09 PM	C File	1 KB
close_enough.exe	2/2/2023 2:09 PM	Application	249 KB
complex.c	3/23/2023 2:31 PM	C File	2 KB
complex.exe	3/23/2023 2:31 PM	Application	411 KB
data.bin	3/24/2023 8:24 AM	BIN File	1 KB
data.txt	3/9/2023 9:03 AM	TXT File	1 KB
dice_roll.c	1/26/2023 11:42 AM	C File	1 KB
dice_roll.exe	1/26/2023 11:42 AM	Application	249 KB
enum.c	2/22/2023 8:58 AM	C File	1 KB
	2/22/2023 8:58 AM	Application	249 KB
	3/9/2023 9:15 AM	C File	1 KB
	3/9/2023 9:15 AM	Application	121 KB
	3/15/2023 9:11 AM	C File	1 KB
	3/15/2023 9:11 AM	Application	249 KB



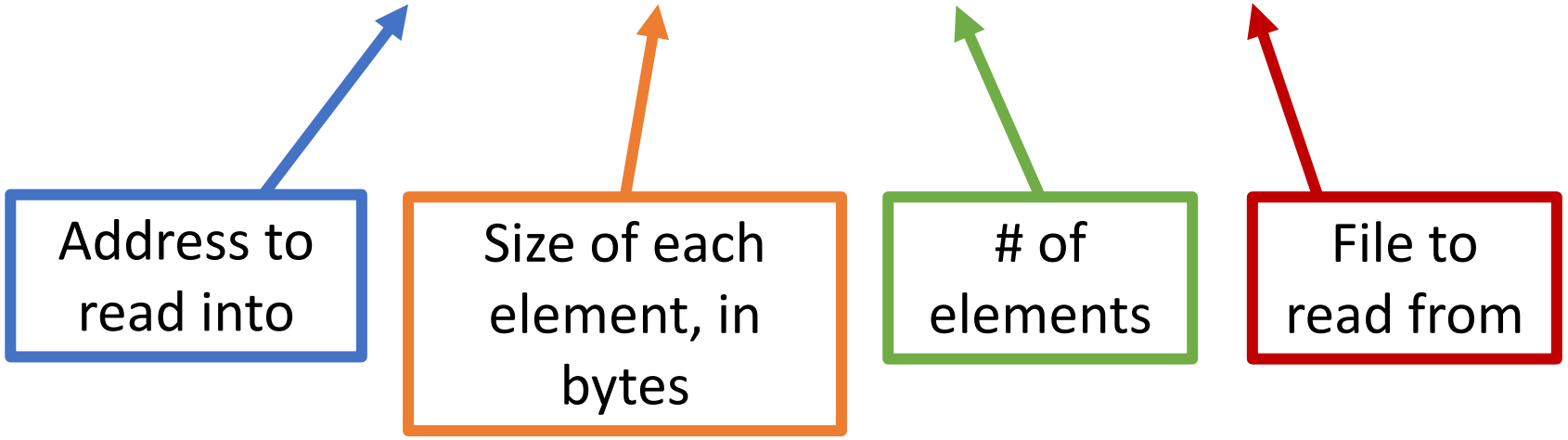
It's not a text file!

- Can't read the contents as ASCII.
- This file contains the two's complement bit pattern for 42

Read Binary, Write Binary

Open it and read as binary:

```
FILE* bin_in;
bin_in = fopen("data.bin", "rb");
int x;
fread(&x, sizeof(x), 1, bin_in);
```



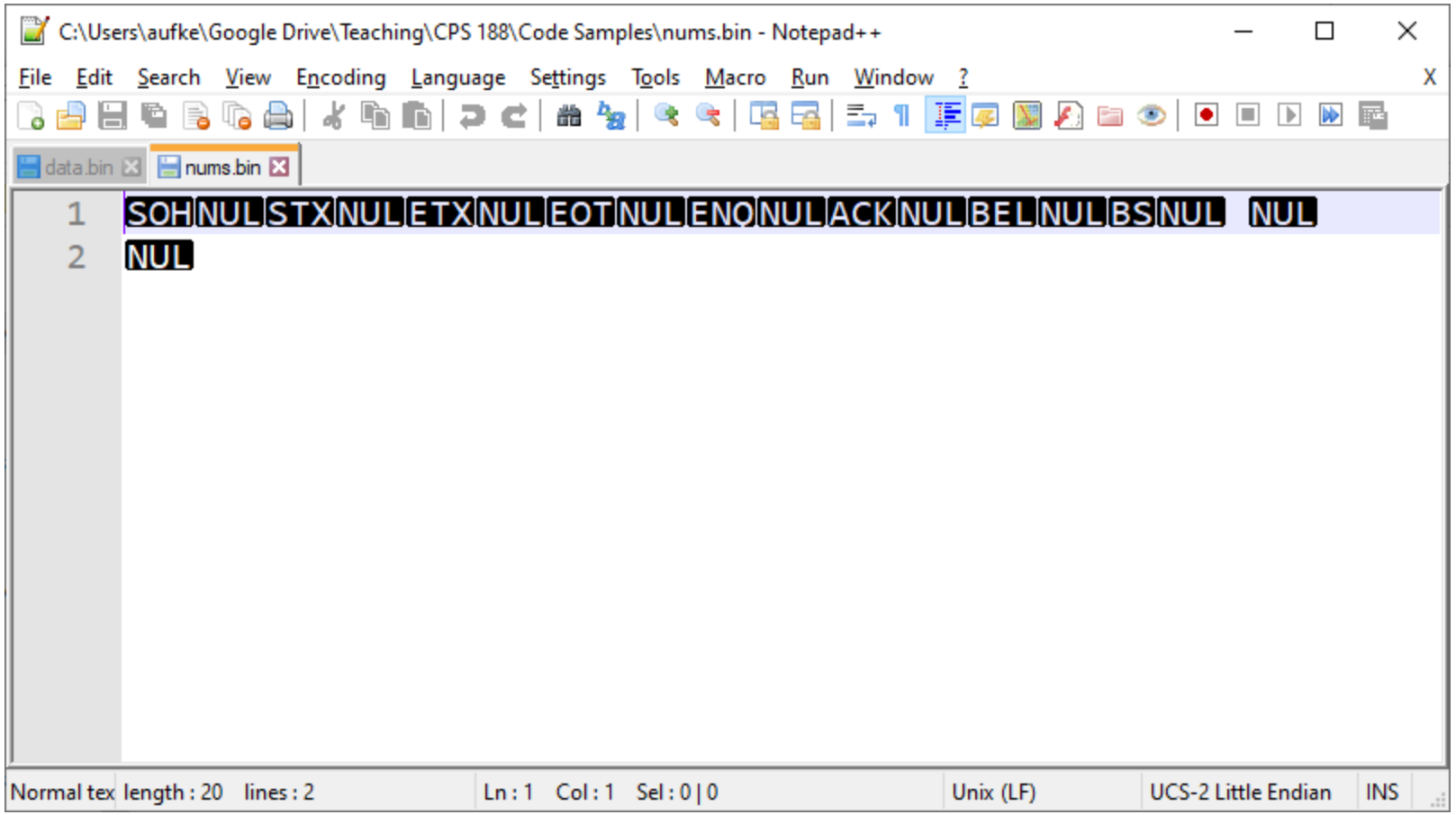
binaryIO.c

```c
#include <stdio.h>

int main (void)
{
    FILE* bin_in = fopen("data.bin", "rb");

    int x;
    fread(&x, sizeof(x), 1, bin_in);

    printf("Read %d\n", x);

    fclose(bin_in);

    return (0);
}
```

A More Powerful Use?

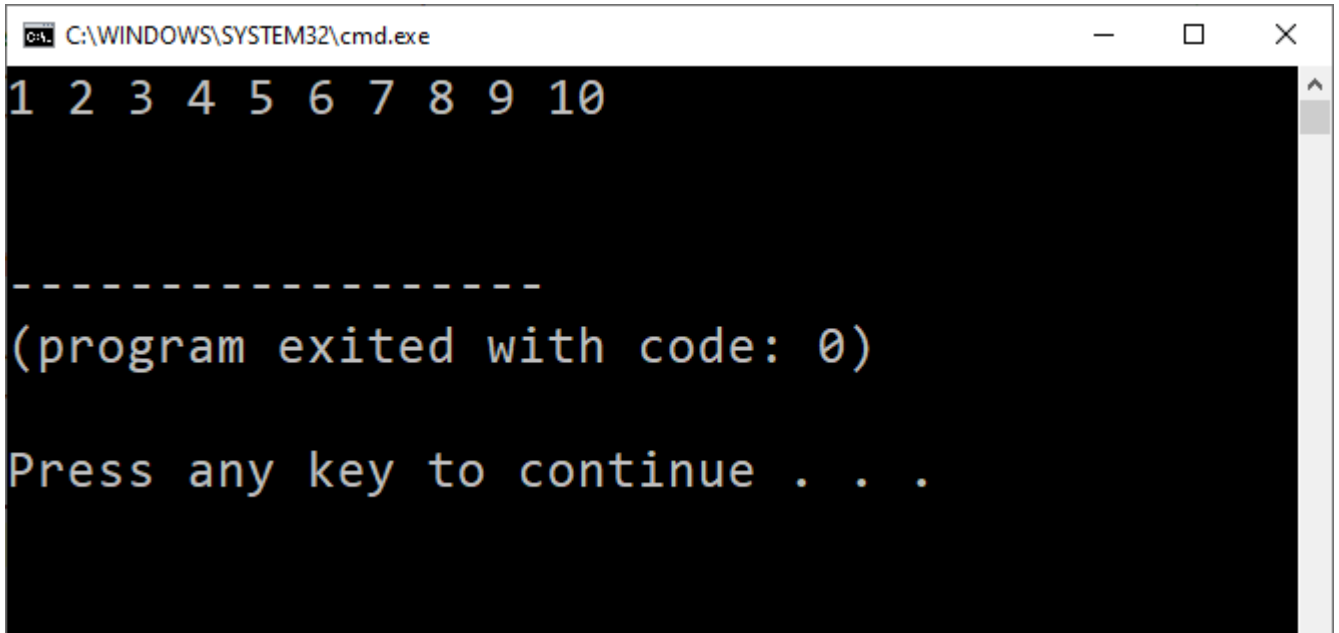
Reading/writing entire arrays in one shot!

```
FILE* bin_out = fopen("nums.bin", "wb");
int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
fwrite(arr, sizeof(int), 10, bin_out);
fclose(bin_out);
```



binaryIO.c

```c
#include <stdio.h>

int main (void)
{
    FILE* bin_out = fopen("nums.bin", "wb");
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    fwrite(arr, sizeof(int), 10, bin_out);
    fclose(bin_out);

    FILE* bin_in = fopen("nums.bin", "rb");
    int arr2[10];
    fread(arr2, sizeof(int), 10, bin_in);

    for (int i = 0; i < 10; i++)
        printf("%d ", arr2[i]);
    printf("\n");

    fclose(bin_out);

    return (0);
}
```



Questions?