CS246
Unix: shell script (conclusion)
C:Huffmann encoding and bits

April 26

Lab

- Write a shell script that sums the size of all writeable files in this directory and the directory immediately below this directory.
- The shell script should not do any directory traversal or listing on its own. Rather all of the file names should be given as command line parameters

- Send me a copy of your script and the unix command line through which this script would be invoked to do the above task.

Lab answer

```
file: blah.sh

#!/bin/bash
TOT=0
for FILE in $@
do
  if [[ -w $FILE  && -f $FILE ]]
  then
    ADET=( `ls -l $FILE` )
    TOT=$(( $TOT + ${ADET[4]} ))
    echo $FILE ${ADET[4]}
    ((CNT++))
  fi
done
echo $TOT
```

COMMANDS:

```
UNIX> chmod 700 blah.sh
UNIX> blah.sh * */*
```

Shell scripts — flagged params

- getopts
  - a shell utility for parsing flagged parameters
  - 2 args to utility‘srd:f:'
    - a list of the flags
      - : indicates the flag has an argument
    - c a variable to hold the arg identity
    - $OPTARG the argument if it exists
  - At end $OPTIND is index of “next” argument

```
usage() {
    echo "usage optargs1.sg [-s] [-r] [
    exit 2
}
if [[ $# -eq 0 ]]
then
    usage
fi
while getopts 'hsrd:f:' c
do
  case $c in
    s) echo "Save"  ;;
    r) echo "Restore" ;;
    d) echo "Drop $OPTARG" ;;
    f) echo "File $OPTARG” ;;
    h) usage
       ;;
  esac
done
shift $((OPTIND – 1))
echo "REST: $*"
```

functions!!!

case == switch

end of “s” handler

end of case

;; can be on same line or next

shift N moves command line param “down” by that number, so “shift 1”
moves $2 into $1 $1 into $0 and forgets the old $0

Real usage of getopts
submit script start

```
while getopts y:c:p:d:i OPT
do
    case $OPT in
        y)     PASSWORD=$OPTARG
               ;;
        c)     COURSE=$OPTARG
               ;;
        p)     PROJECT=$OPTARG
               ;;
        d)     TARGET=$OPTARG
               ;;
        i)     INIT=1
               ;;
        esac
done

shift $((OPTIND – 1))
```

getopts is restrictive

- flags must be single chars
- flags must have exactly 0 or exactly one arg
    - not 0 or 1
- No handling of poor user input
- no flags preceded by - -
- No flags with 2 or more args

When getopts is too restrictive Write your own!

```
while [[ -n "$1" ]]
do
    # make args the same as previous optargs
    case "$1" in
      -s|-save) echo "Save"
            shift  ;;
      -r|--restore)
            regexInt='^[0-9]+$'
            if [[ $2 =~ $regexInt ]]; then
                echo "restore $2”;  shift 2
            else
                echo “restore”; shift
                fi ;;
      --t|--triple)
          echo "triple $2 $3"
          shift 3
          ;;
      *)
          # On any unknown arg exit the while loop so those
          # can be processed elsewhere
          break
          ;;
    esac
done
echo "AAA $*"
```

optargs strips off the leading “-“. This does not

optargs implicitly shifts inside the while loop

optional integer argument – uses regex to recognize integer and “=~” to mean “looks like”

separates values that do take the same action

Takes 2 args!

Anything else!!

Shell scripts and my huffman code

- When writing my code for this assignment I considered a different interface.
- Rather than rewriting C, I wrapped my C executables in a shell script that has the exact interface of the assignment!

```
file: huffmann
(/home/gtowell/Public/246/HW8)


#!/bin/bash
if [[ $# –ne 3 ]]
then
    echo "usage huffmann [1 or 2] inf
else
    if [[ $1 –eq 1 ]]
    then
        huff $2 –1 > $3 2> /dev/null
    else
        puff < $2 > $3
    fi
fi
```

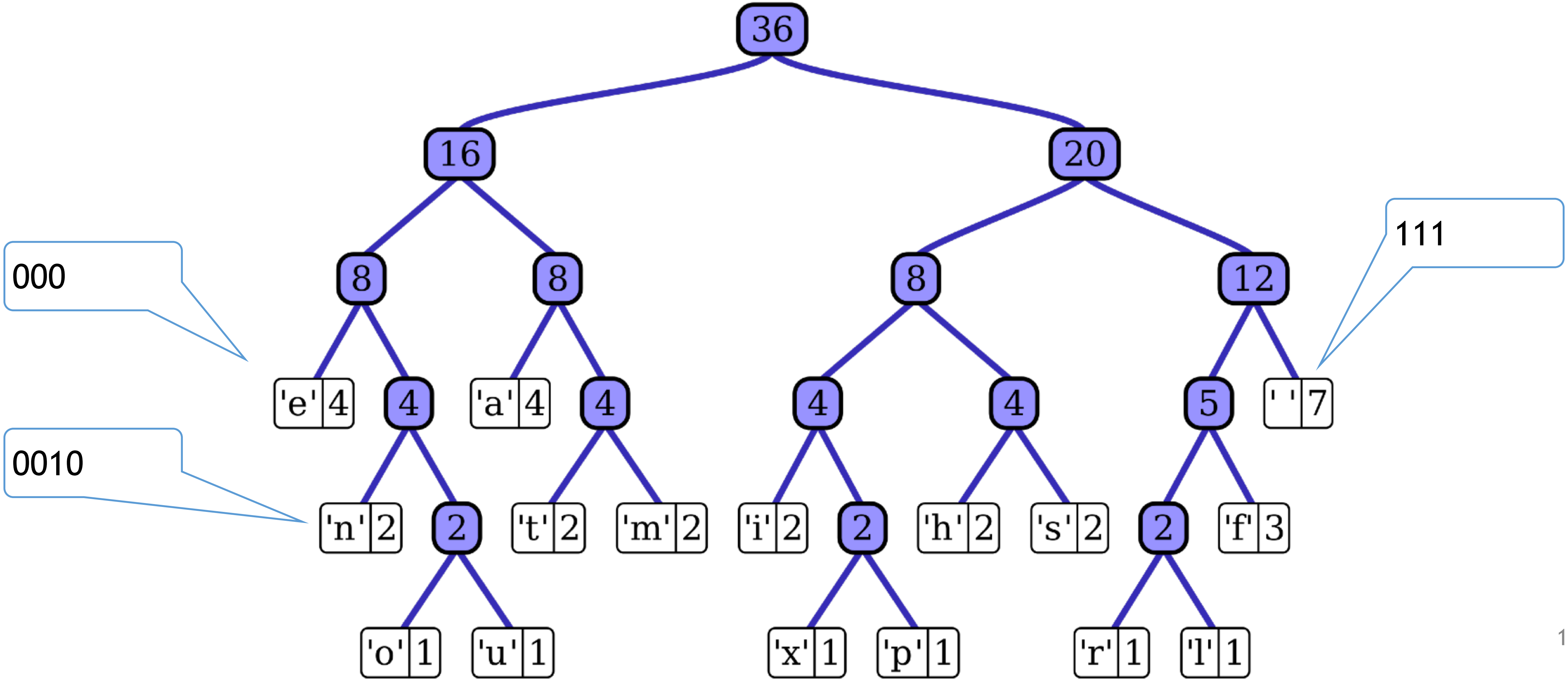
Huffman Coding

- Assign codes to characters such that the length of the code depends on the relative frequency of the character
- Codes are this of variable length
- Huffman codes are “prefix free”
    - No code is the prefix of any other.
        - Example 1
            - suppose the code for e is 1
            - then every other code MUST start with 0.
            - further suppose the code for a is 00
            - then every other code must start with 01
        - Example 2
            - suppose the code for e is 1, the code for a is 0
            - There can be no other codes!!!

David Huffman, 1952
(1925-1999)
while working on PhD at MIT

Prefix free binary trees

- Can be visualized as a binary tree with letters at leaves



Building a Huffman coding tree

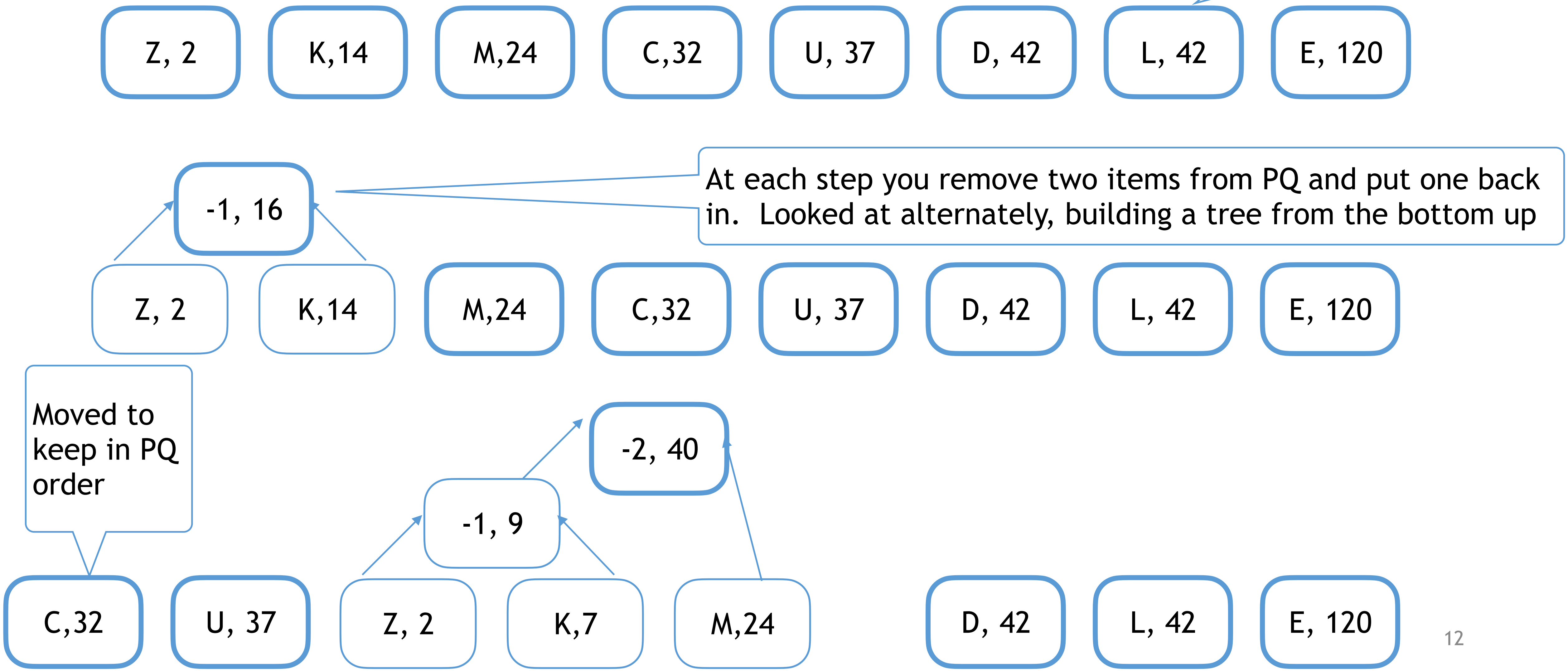
Letter	Z	K	M	C	U	D	L	E
Frequency	2	14	24	32	37	42	42	120

This is the priority queue!

- Idea — build a tree from the bottom up by progressively merging the least frequent items
    - think of the letters as existing in a priority queue where the keys is the frequency and ties are broken in favor of the smaller letter
- Suppose characters and frequencies are as above.
- Algorithm:
    - remove two items from priority queue.
    - create a new item that is the merger of the two removed.
    - Add new item to PQ
- So the first step is the merge Z and K

Tree building, graphically

Thick borders indicate items in PQ



Z, 2 K,14 M,24 C,32 U, 37 D, 42 L, 42 E, 120

-1, 16

At each step you remove two items from PQ and put one back in. Looked at alternately, building a tree from the bottom up

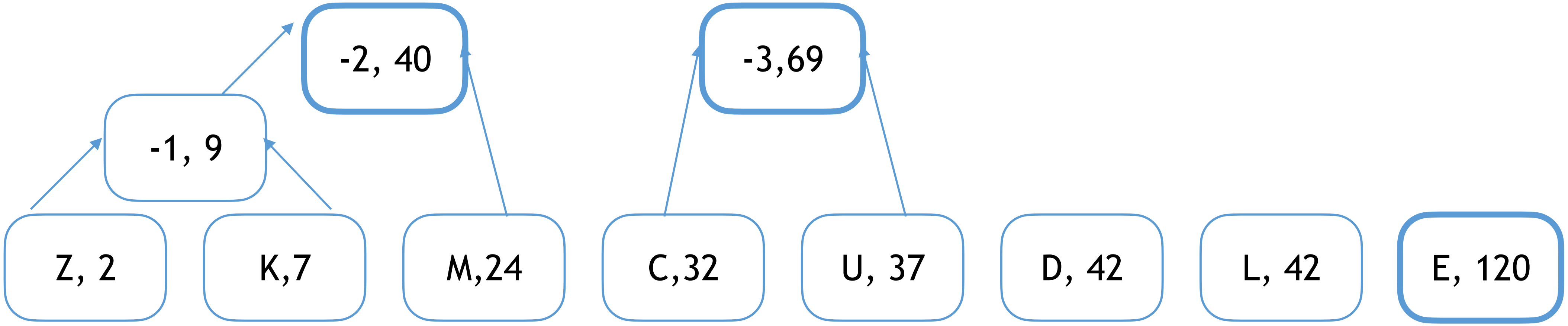
Z, 2 K,14 M,24 C,32 U, 37 D, 42 L, 42 E, 120

Moved to keep in PQ order

-2, 40

-1, 9

C,32 U, 37 Z, 2 K,7 M,24 D, 42 L, 42 E, 120



-2, 40

-3,69

-1, 9

Z, 2

K,7

M,24

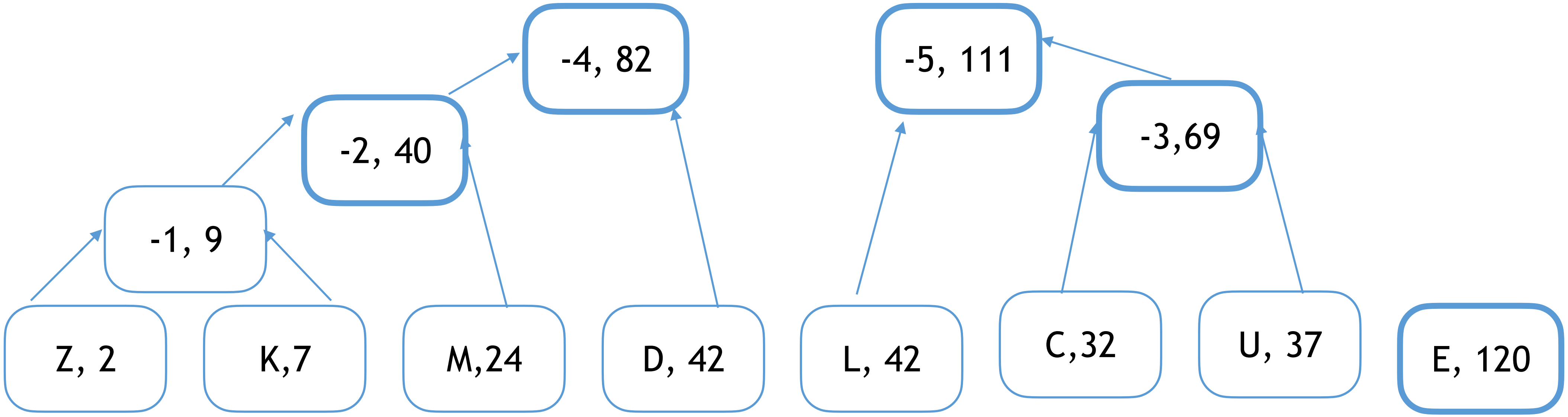
C,32

U, 37

D, 42

L, 42

E, 120



-4, 82

-5, 111

-2, 40

-3,69

-1, 9

Z, 2

K,7

M,24

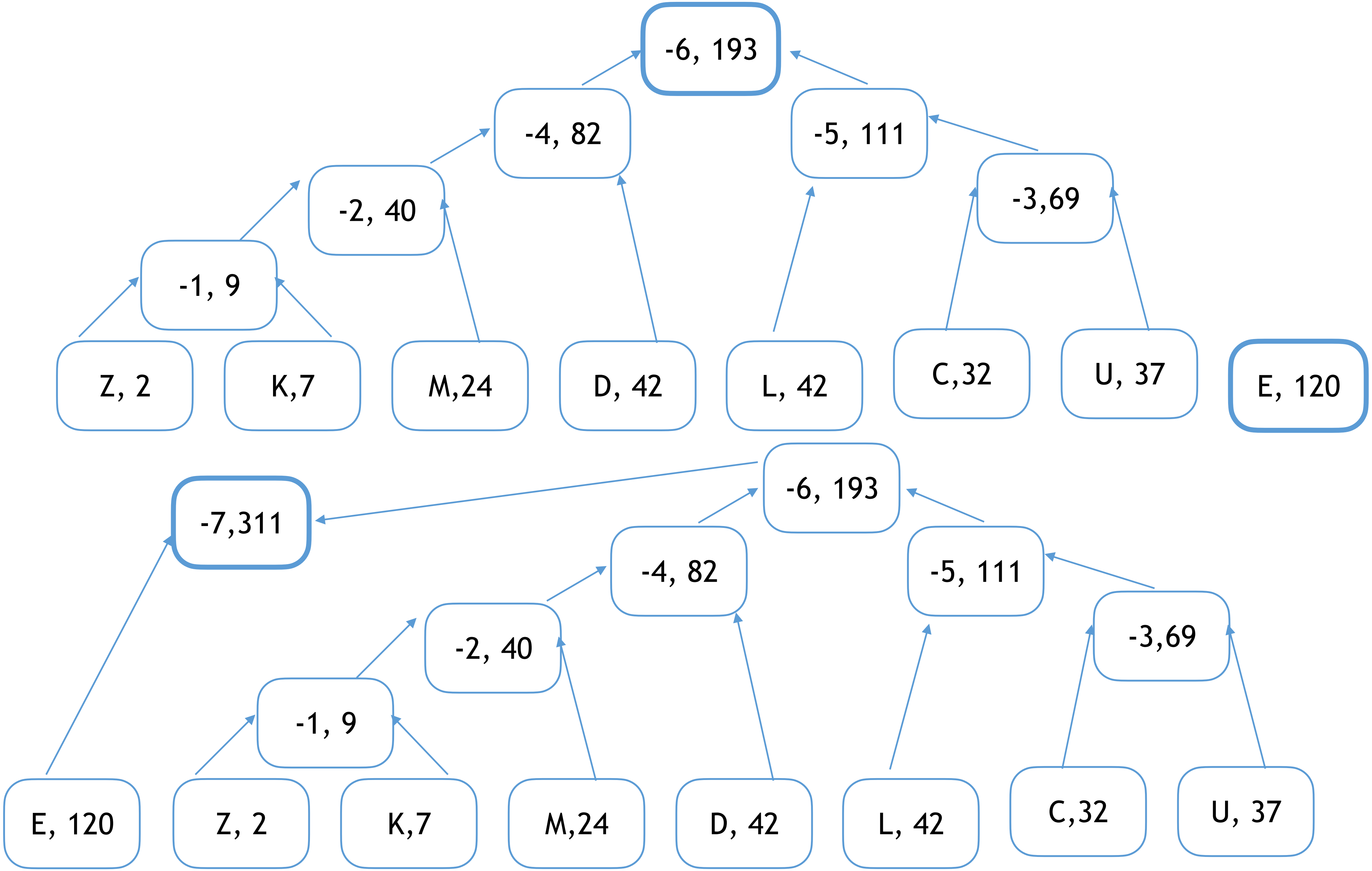
D, 42

L, 42

C,32

U, 37

E, 120



Only one i
so the tree

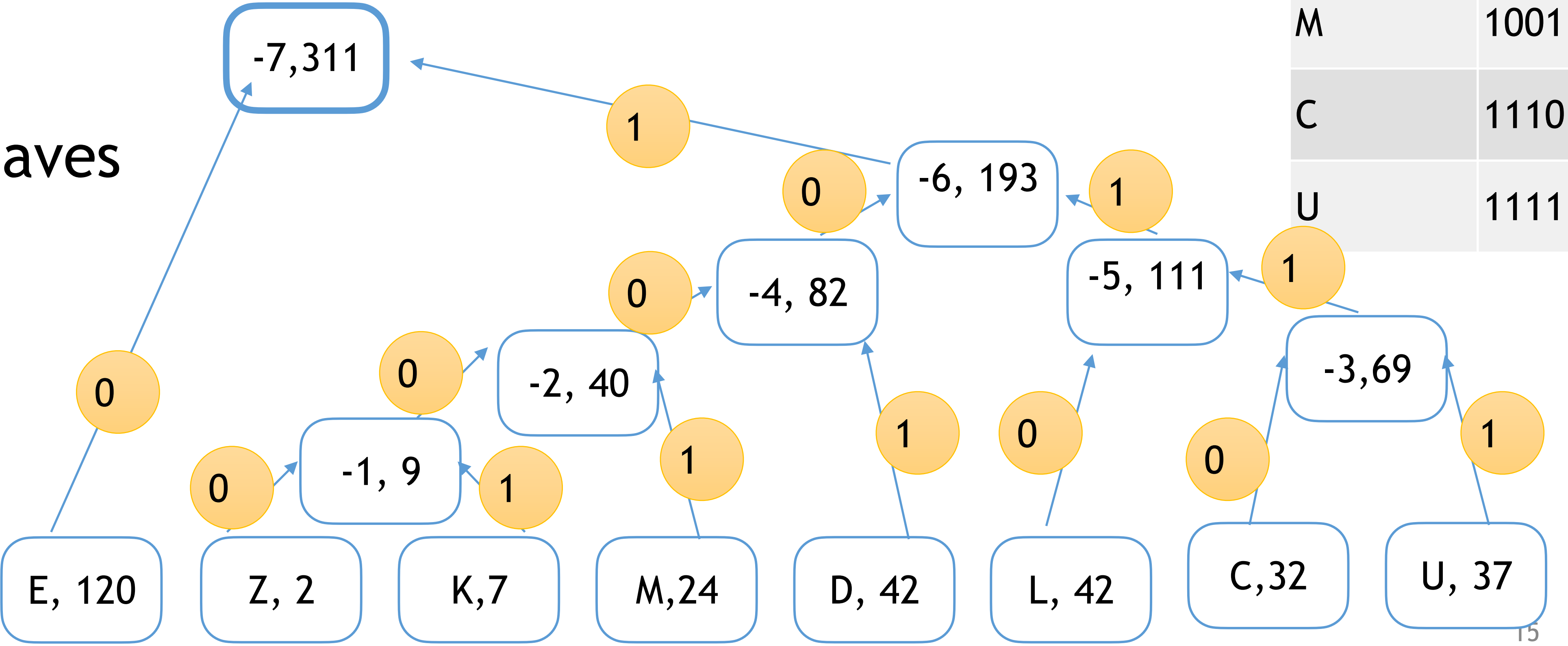
Reading the Huffman Tree

E	0
D	101
L	110
Z	10000
K	10001
M	1001
C	1110
U	1111

10011111111101100101

- left branch=0
- right branch=1

- read down paths to leaves to get the encoding



Huffman Encoding

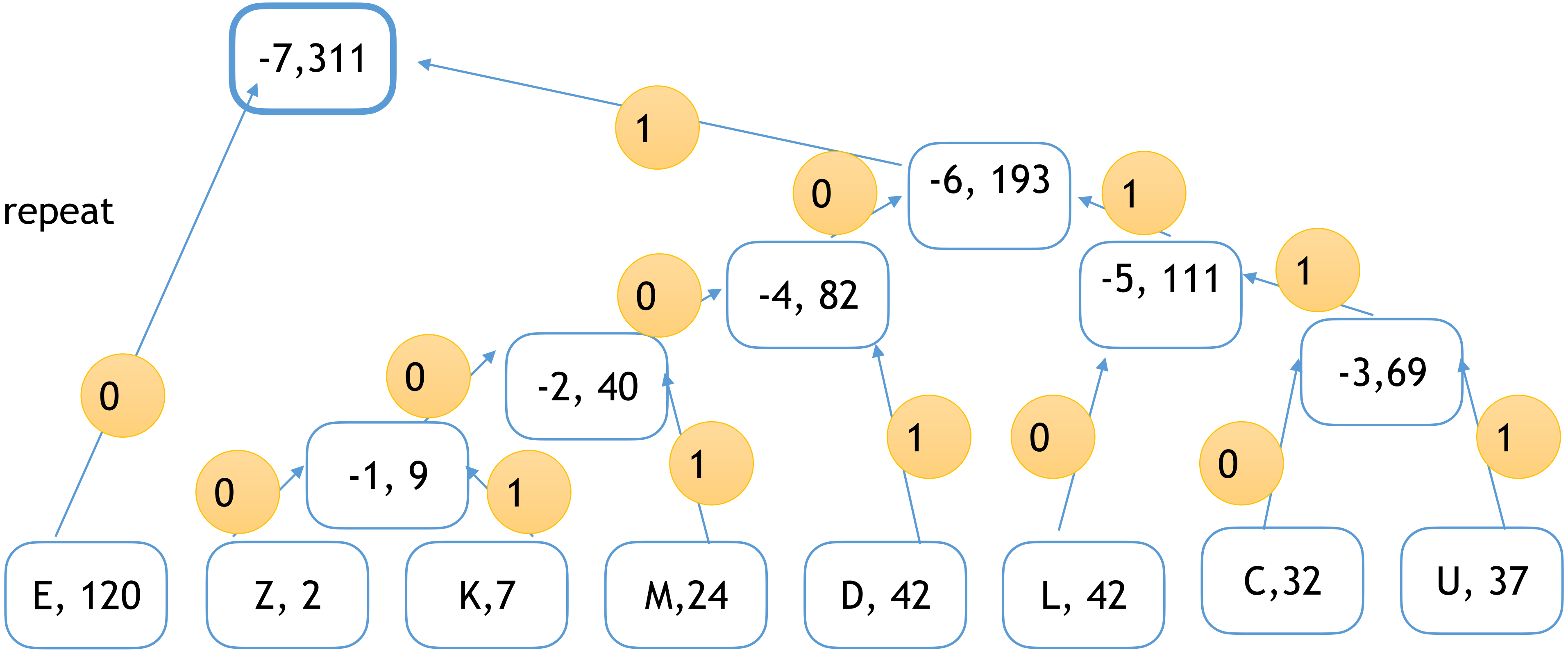
- Use tree to generate table
- just substitute codes for letters

- CUED = 1110111101101

E	0
D	101
L	110
Z	10000
K	10001
M	1001
C	1110
U	1111

Huffman Decoding

- Use the tree.
- Navigate by 0=left 1=right
- if at a left, output letter,
    - return to the top and repeat
- 1001111111101100101
- left, right, right, left
    - output M
- left, left, left, left
    - output U
- left, left, right ==> L
- left, left, right ==> L
- right ==> E
- left, right, left => D



- Problem, in compressed file you do not have the characters so cannot build tree!!
    - solution: include the character counts in compressed file

Huffman Optimality

- Huffman codes are the best prefix-free encoding
    - proof derived from “weights”

- But gzip makes smaller files!!!
-

char	code	bits in code	freq	weight
E	0	1	120	120
D	101	3	42	126
L	110	3	42	126
Z	10000	5	2	10
K	10001	5	14	70
M	1001	4	24	96
C	1110	4	32	128
U	1111	4	37	148

Saving Huffman Compressed Files

- from previous slide 100111111101100101 decompresses to MUDDLE.
    - but 100111111101100101 is 18 characters and MUDDLE is only 6!!!
    - Worse you can also consider it to be 16 integers!

- BUT, you can also consider it to be 18 bits!

- Need to read and write bites/bytes
    - 8 bits per byte, so if store 110111111011010100 as bits then only need 3 bytes.
        - COMPRESSION!!!!

fread and fwrite

- `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
    - ptr – the things to write
    - size – the size of the things to be written
    - nmenb – the number of things to be written
    - stream – the place to write
    - return – the number of bytes written

Order of these is reversed from qsort!

- `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`

fwrite/fread

- advantages over fprintf
  - faster & simpler
  - flexibility — can write literally anything (big ugly structs)
  - Size. Writing bits so typically smaller than printer representation (int is 32 bits, print rep may be 16 chars …)
  - No annoying casting / conversion
  - Write/read a struct (or array of structs) in 1 line!!!!
- Disadvantages
  - handling pointers
  - Fragility
    - you have to know exactly how (and what) the file was written to read

Save/read int and double using fprintf and fscanf

```
int main(int argc, char const *argv[])
{
    int a = 5;
    double b = 10.854897548357;
    FILE *fp = fopen("lab18", "w");
    fprintf(fp, "%4d %12.8f\n", a, b);
    fclose(fp);
    return 0;
}
```

```
int main(int argc, char const *argv[])
{
    int a;
    double b;
    FILE *fp = fopen("lab18a", "r");
    fscanf(fp, "%d %lf", &a, &b);
    fclose(fp);
    printf("%d %f\n", a, b);
    return 0;
}
```

NOTE: using fscanf to read a file you wrote by fprintf is only time to use fscanf (IMHO)

Save/read int and double
fwrite / fread

```c
int main(int argc, char const *argv[])
{
    int a = 5;
    double b = 10.854897548357;
    FILE *fp = fopen("lab18b", "w");
    fwrite(&a, sizeof(int), 1, fp);
    fwrite(&b, sizeof(double), 1, fp);
    fclose(fp);
    return 0;
}
```

```c
int main(int argc, char const *argv[])
{
    int a;
    double b;
    FILE *fp = fopen("lab18b", "r");
    fread(&a, sizeof(int), 1, fp);
    fread(&b, sizeof(double), 1, fp);
    fclose(fp);
    printf("%d %f\n", a, b);
    return 0;
}
```

write
array of structs

- compile
  - gcc -lm xxx.c
  - -lm == “include math library”
  - math library is not part of standard C library
  - in /usr/lib/libc.m

```
#include <stdio.h>
#include <math.h>
#define SIZ 100
typedef struct
{
    int aa;
    double bb;
} aabb;

int main(int argc, char const *argv[])
{
    aabb arr[SIZ];
    for (int i = 0; i < SIZ; i++) {
        arr[i].aa = i;
        arr[i].bb = sqrt(i);
    }
    FILE *fp = fopen("astr", "w");
    fwrite(arr, sizeof(aabb), SIZ, fp);
    fclose(fp);
    return 0;
}
```

read array of structs

- Fails horribly at runtime!!
- WHY?
- Link to Java objet serialization
  - serialVersionUID
- Also SIZ

```c
#include <stdio.h>
#define SIZ 100
typedef struct {
    double bb;
    int aa;
} aabb;

int main(int argc, char const *argv[])
{
    aabb arr[SIZ];
    FILE *fp = fopen("astr", "r");
    fread(arr, sizeof(aabb), SIZ, fp);
    fclose(fp);
    for (int i = 0; i < SIZ; i++)
        printf("%d %f\n", arr[i].aa, arr[i].bb);
    return 0;
}
```

fwrite — problems

- arrays of pointers!!

- What will the file contain?
- How big will the file be?

```
#define SIZ 100
typedef struct
{
    int aa;
    double bb;
} aabb;

int main(int argc, char const *argv[])
{
    aabb* arr[SIZ];
    for (int i = 0; i < SIZ; i++) {
        arr[i] = malloc(1 * sizeof(aabb));
        arr[i]->aa = i;
        arr[i]->bb = sqrt(i);
    }
    FILE *fp = fopen("astrfp", "w");
    fwrite(arr, sizeof(aabb), SIZ, fp);
    fclose(fp);
    return 0;
}
```

fwrite fixed

- Why not write the array of pointers??
- Problems remain??

```c
#define SIZ 100
typedef struct
{
    int aa;
    double bb;
} aabb;
int main(int argc, char const *argv[])
{
    aabb **arr = malloc(SIZ * sizeof(aabb *));
    for (int i = 0; i < SIZ; i++) {
        arr[i] = malloc(1 * sizeof(aabb));
        arr[i]->aa = i;
        arr[i]->bb = sqrt(i);
    }
    FILE *fp = fopen("astrfp", "w");
    int d = SIZ;
    fwrite(&d, sizeof(int), 1, fp);
    for (int i = 0; i < SIZ; i++)
        fwrite(arr[i], sizeof(aabb), 1, fp);
    fclose(fp);
    return 0;
}
```

freading what you fwrote

- Read in exactly the order you wrote

- It works!!!!

```c
typedef struct
{
    int aa;
    double bb;
} aabb;

int main(int argc, char const *argv[])
{
    aabb **arr;
    FILE *fp = fopen("astrfp", "r");
    int d;
    fread(&d, sizeof(int), 1, fp);
    arr = malloc(d * sizeof(aabb *));
    for (int i = 0; i < d; i++) {
        arr[i] = malloc(1 * sizeof(aabb));
        fread(arr[i], sizeof(aabb), 1, fp);
    }
    fclose(fp);
    for (int i = 0; i < d; i++)
        printf("%d %f\n", arr[i]->aa, arr[i]->bb);
    return 0;
}
```

Lab

- Given the table at right
  - Construct a Huffman tree
  - Encode
    - abed
  - Decode
    - 101100110011010

- Send: tree, encoding and decoding

Character	Count
a	2
b	5
c	14
d	14
e	7