CS246
Unix:Scripts
C:more structs

March 22

Lab fromTuesday

```c
typedef struct{
    int num;
    char c;
} pType;

void printPT(pType * p){
    printf("%d, %c\n", p->num, p->c);
}

void initter(int siz, pType ppp[siz] ) {
    for(int i = 0; i < siz; i++){
        ppp[i].num = rand()%100;// random integer from 0 to 99
        ppp[i].c = (char)(rand()%26+'a');// random lowercase letter
    }
}

int main(int argc, char const *argv[]) {
    srand(time(NULL));
    pType arr[SIZ];
    initter(SIZ, arr);
    for (int i = 0; i < SIZ; i++) {
        printPT(&arr[i]);
    }
    return 0;
}
```

Problem

- Given a text file, find the N most common words, where N is input from the command line
    - optimize for your time to write
        - fewest lines of C code possible

- Requirements:
    - case insensitive
    - handles punctuation by ignoring/eliminating it

Can I use UNIX functions to make the task easier??

- Case Insensitive
  - tr A-Z a-z
- Get rid of all punctiation
  - tr '\?!,\.-' ' '   works for green eggs and ham
  - tr '[:punct:]' ' ‘   works for all text
- put one word on each line — no strtok in C
  - tr ' ' ‘\n’
- finally group all like words — no search in C
  - sort

Together and in a file

- tr [:punct:] ' ' | tr A-Z a-z | tr ' ' '\n' | sort

- precede with
- #!/bin/bash

- cat $1 | …
    - the $1 indicates the first command line argument
- chmod 777 ppp.sh

```
#!/bin/bash
cat $1 | tr [:punct:] ' ' | tr A-Z a-z | tr ' ' '\n' | sort
```

```
./ppp.sh ham.txt | uniq | wc
     51      50     223
```

```
time ./ppp.sh janeausten.txt | uniq | wc
  20312   20310  184440

real	0m1.817s
user	0m1.870s
sys	0m0.062s
```

Now to C

- Count number of times a word appears
    - since words appear consecutively, that is pretty easy
- Once you know count for current word, find the current least frequent word.
- If current more frequent than least frequent, replace least freq with current

```
typedef struct {
    char word[50];
    int count;
} Words;


int main(int argc, char* argv[]) {
    int numMax=DEFAULT_MAX;
    if (argc>1) {
        numMax =atoi(argv[1]);
    }

    Words words[numMax];
    for (int i=0; i<numMax; i++)
        words[i].count=-1;

    doCounting(numMax, words);

    for (int i=0; i<numMax; i++)
        wprinter(&words[i]);
}
```

More C

- Count number of times a word appears
- If word changes, try to add the former word to the most common list and start new word
-

```
void doCounting(int arrSize, Words words[arrSize]) {
    char instring[50];
    char curstring[50];
    int ccount = 0;
    while (fgets(instring, 50, stdin)) {
        //printf("%s\n", instring);
        if (0 == strcmp(instring, curstring)) {
            ccount++;
        }
        else {
            addBig(arrSize, words, ccount, curstring);
            ccount=1;
            strcpy(curstring, instring);
        }
    }
    // handle the last word
    addBig(arrSize, words, ccount, curstring);
}
```

Finally

- so to use
  - gcc -o counter countn.c
  - ./ppp.sh xxx.txt | counter 10
  -

```
int smallestIndex(int count, Words words[count]) {
    int lowi=0;
    int lowc=words[0].count;
    for (int i=1; i<count; i++) {
      if (words[i].count<lowc) {
        lowc=words[i].count;
        lowi=i;
    }}
    return lowi;
}

void addBig(int count, Words words[count], int ccount, char* cu
    int lowi = smallestIndex(count, words);
    if (words[lowi].count<ccount) {
        strcpy(words[lowi].word, curstring);
        words[lowi].count=ccount;
    }}

void wprinter(Words* ww) {
    if (ww->count>0)
      printf("%d %s", ww->count, ww->word);
}
```

Better Yet

- The step I really had to write code for was the counting the number of occurrences of a word.
- Code did that AND
  - collected up top N
- Use UNIX to collect up top N

- Idea.
  - C code just outputs word and count
  - sort that
  - tail (or head depending on sort direction)

- Result
  - cut about 60% of code from countn.c
- Extend ppp.sh into ppp2.sh
  - 7!! pipes

```
file: ppp2.sh

#!/bin/bash
cat $1 | tr [:punct:] ' ' | tr A-Z a-z |
tr ' ' '\n' | sort | count2 | sort -n -k
2 | tail -n $2
```

Structs within structs

- like Java classes you can put a struct within a struct
- For weather data
  - structs for each component

leave space for \0

The entire weather data struct

```c
typedef struct {
    char time[10];
    char ampm[3];
} Time;
typedef struct {
    char direction[10];
    int speed;
    char scale[4];
} Wind;

typedef struct {
    int temperature;
    char scale[3];
} Temperature;

typedef struct {
    Time time;
    Temperature temperature;
    Temperature dewPoint;
    int relHum;
    Wind wind;
} WeatherData;
```

printing and sprintf

- String printf
  - does like printf, but puts result into a string.
- realistically, should always use snprintf
  - like sprintf but respects space
- so now have
  - printf
  - fprintf
  - snprintf / sprintf
- Java equivalent??

```
void t(int ii) {
    char aaa[10];
    sprintf(aaa, "t %d", ii);
}

void tn(int ii) {
    char aaa[10];
    snprintf(aaa, 10, ”t %d", ii);
}
```

Splitting weather

- “Best practice” is to take the java route and have separate files for each struct
  - put the struct in .h
  - put struct specific functions in .c
- Private functions!!!
  - by not including functions in .h file they are, effectively, private

```c
#include "wtime.h"
#include "wtemp.h"
#include "wwind.h"

typedef struct {
    Time time;
    Temperature temperature;
    Temperature dewPoint;
    int relHum;
    Wind wind;
} WeatherData;

extern WeatherData weather[];

void wprinter(WeatherData *w);
int readFile(char *fileName);
```

wweather.c

- parse is largely unchanged from parseC
    - but it is now private!!

```
#include "wweather.h"
void wprinter(WeatherData* w) {
    char timee[55];
    timeToString(timee, &(w->time));
    char tempe[55];
    tempToString(tempe, &(w->temperature));
    printf("Time:%s  Temp:%s\n", timee, tempe);
}
void parse(char* line, WeatherData * w){
}
int readFile(char* fileName) {
    char line[256];
    FILE* f = fopen(fileName, "r");
    if (f==NULL) {
        fprintf(stderr, "Could not open %s -- quitting\n", fileN
        return -1;
    }
    int c = 0;
    while (NULL != fgets(line, 256, f))
    {
        parse(line, &weather[c]);
        c++;
    }
    return c;
}
```

More Splitting

- time shown
- similar for temp and wind

```
file: wtime.h
typedef struct {
    char time[10];
    char ampm[3];
} Time;
void timeToString(char *target, Time *time);
```

```
file: wtime.c

#include <stdio.h>
#include "wtime.h"

void timeToString(char* target, Time* time)
{
    sprintf(target, "%s %s", time->time, time->ampm);
}
```

Why not snprintf??

Main after split

- Not a whole lot left

```
#include "wweather.h"


WeatherData weather[100];

int main(void)
{

    int count = readFile("temps.txt");
    for (int i=0; i<count; i++)
    {
        wprinter(&weather[i]);
    }
}
```

return of makefile

- before splitting
  - 1 file with 96 line
- after splitting
  - 8 files 126 lines

- Building across 8 files is cumbersome

```
wweather: wmain.c wweather.o wtemp.o
wwind.h wtime.o
    gcc –o wweather wmain.c wweather.o
wtemp.o wtime.o

wweather.o: wweather.c wweather.h
    gcc –c wweather.c

wtime.o: wtime.c wtime.h
    gcc –c wtime.c

wtemp.o: wtemp.c wtemp.h
    gcc –c wtemp.c

clean:
    rm *.o
```

Trimming Strings
Contant strings
Immutable Strings

- Problem: remove unwanted characters from the beginning and end of a string
    - Usually whitespace ‘ ‘ or ‘\t’ but could be anything.

- char* trim(char* target, char*charsToTrim)

- How:
    - front of string:
    - back of string:

Trim at the front

- do not change the string at all
- Just return the place where the trimmed string should begin

```
char* ftrim(char* target, char* toBeTrimmed)
{
    int ff=1;
    while (ff)
    {
        char *tbt = toBeTrimmed;
        ff = 0;
        while (*tbt !='\0') {
            if (*tbt == *target) {
                ff = 1;
                target++;
                break;
            }
            tbt++;
        }
    }
    return target;
  }
```

trim the back

- start at the end and work backward, replace all whitespace with \0
- You are changing the string.
-

```
char* trim(char* target, char* toBeTrimmed) {
    int ff = 1;
    char* trailer = target;
    while ('\0' != *trailer) { trailer++; }
    if (trailer!=target) {
        trailer--;
        ff = 1;
        while (ff) {
            ff = 0;
            char *tbt = toBeTrimmed;
            while (*tbt !='\0') {
                if (*tbt == *trailer) {
                    *trailer = '\0';
                    trailer--;
                    ff = 1;
                    break;
                }
                tbt++;
            }}}
    return target;
}
```

Trim

- Depending on how they are set up strings in C may be a part of the compiled code and immutable
- Or in heap/stack memory and so mutable

```
int main(void) {
    // Strings defined like this are immutable!!
    // this literally get compiled into the program
    // and are a part of the program image
    // front trim still works because the string is not ch
    // only the pointer to the front is moved
    trimTest("   this is a test");
    // Strings like this are mutable!!
    // they are a part of the stack space
    // See diff in memory location
    char str[] = "  test   ";
    trimTest(str);
    // this is immutable also!! Again, part of the program
    // then the program dies because to trim the back need
    // at the end of the string.
    char *strP = "  asdg   ";
    trimTest(strP);
    trimTest("\tThird");
}
```

valgrind

- So my program died with a seg fault … at least tell me where
- Step 1: compile with debug flags
  - gcc -g trim.c
- Step 2 use valgrind
  - valgrind [-v] ./a.out
    - the -v is for “verbose”
- This will at least tell you the exact line where your program died

```
[gtowell@powerpuff L11]$ gcc -g trim.c
[gtowell@powerpuff L11]$ valgrind a.out
==532493== Memcheck, a memory error detector
==532493== Copyright (C) 2002-2017, and GNU GPL'd, by Julian
==532493== Using Valgrind-3.15.0 and LibVEX; rerun with -h for
==532493== Command: a.out
==532493==
ORIGINAL     1089609 ||   this is a test||
TRIMMED      1089612 ||this is a test||
ORIGINAL   -16774786 ||  test   ||
TRIMMED    -16774784 ||test||
ORIGINAL     1089627 ||  asdg   ||
==532493==
==532493== Process terminating with default action of signal 11
==532493==  Bad permissions for mapped region at address 0x10
==532493==    at 0x10920E: trim (trim.c:49)
==532493==    by 0x109275: trimTest (trim.c:69)
==532493==    by 0x109308: main (trim.c:94)
==532493==
```

NO LAB