CS246
Unix: grep
C: pass by value, references

March 4
grep

Global Regular Expression Print

• One of the most used Unix utilities
• Idea: from standard input (or file) find lines that contain a “regular expression”
  • or just a string
• Example
  • LS -R – recursively list all files
  • ls -R | grep c
    • finds all files with the letter c
  • grep Darcy ~/public/206/a4/janeausten.txt
    • find all lines that contain “Copperfield” in my dickens collection
    • really long so
      • grep Darcy ~/public/206/a4/janeausten.txt | wc
the RE part of gREp

• Regular expression
  • a way of allowing for broader classes of matches
    • Anchors
      • ^ the beginning of a line
        • show only directories in ls
          • ls -l | grep ^d
      • $ the end of the line
        • show all files in ls that end in s
          • ls -l | grep s$
the RE part of gREp

• . — any single character
  • find all lines containing d, two characters, y
    • grep “d..y” Public/206/a4/janeausten.txt
• [] a character group — match to any single character in group
  • find all lines containing d, a vowel, y
    • grep “d[aeiou]y” Public/206/a4/janeausten.txt
  • find all lines containing d, a letter, y
    • grep “d[a-z]y” ...
• Same but case insensitive
  • grep “[dD][a-zA-Z][yY]” ...
  • grep -i “d[a-z]y” ...
the RE part of gREp

• Quantifiers
  • Apply to the previous character (or group)
  • * — match to 0 or more
    • .* == match to 0 or more occurrences of any letter
      • d.*y matches dy, day, dly, d_y, duly, daddy, ...
  • ? — 0 or 1
    • a? == match to a string that has 0 or 1 a
      • da?y matches dy, day
  • + — 1 or more
    • [a-z]+ one or more instances of any lower case letter
      • d[a-z]+y matches day,dly, daddy, ...
grep — escapes and quotes

• suppose you want to find a line containing . *, or +, or [, or any other character used specially in regular expressions
  • precede that char with \n    • sometimes called the “escape character”
  • Find all lines containing the character “.”
    • grep “\.” dickens.txt
• It is often important — and never wrong — to put REs in quotes
  • grep “\.” dickens.txt — lines containing a .
  • grep \. dickens.txt — every line in the file
• without quotes characters can get interpreted by the shell
  • grep * dickens.txt
    • the * is interpreted by the shell to be a filename expansion operator
      • e.g. grep dickens *.txt
LAB from Monday

• Write your own implementation of strcpy

  • `void strcpy(int destLen, char dest[destLen], char source[]);`

```c
void strcpyGT(int ll, char tgt[ll], char src[]) {
    int i = 0;
    for (; i < ll - 1 && src[i] != '\0'; i++) {
        tgt[i] = src[i];
    }
    tgt[i] = '\0';
}
int main(int argc, char const *argv[]) {
    char line[LINELLEN];
    while (fgets(line, LINELLEN, stdin) != NULL) {
        char copy[LINELLEN];
        for (int i = 0; i < LINELLEN; i++) copy[i] = 'z';
        strcpyGT(LINELLEN, copy, line);
        printf("%d %d %s %s
", strlen(line), strlen(copy), line, copy);
    }
    return 0;
}
```

What happens without this???
Homework 3

• posted on class website
• timing – see code in timer.c for today’s lecture for 3(!) different ways of timing
Pass by value vs Pass by Reference

- Function Calls
  - Pass by value
    - make a copy and work with that
    - changes inside function do not affect outside
  - Pass by reference
    - Work with the same exact thing
    - Change inside function change the outside
PbV or PbR

• Which
  • Java
    • PbV on primitive types
    • PbR on objects
  • C
    • PbV on basically everything
    • BUT there is an catch
PbV or PbR

• Why do I care
  • The effect of changing values in functions
    • javascript “vars” are effectively PbR
  • Speed & memory
    • PbR faster and more memory efficient
  • PbV “safer”?
    • NO side effect programming
& operator

- the “address” operator
  - The memory address of the variable
  - Using & can really observe PbV in action

- Program at right one global variable and a function with no args
  - What is the output?

```c
#include <stdio.h>

int gi = 5;

void t()
{
    printf("TF    %d  %d\n", gi, &gi);
    gi = 7;
    printf("TF2   %d  %d\n", gi, &gi);
    return;
}

int main(void)
{
    printf("TM    %d  %d\n", gi, &gi);
    t();
    printf("TM2  %d  %d\n", gi, &gi);
}
```

Show the address in memory as an integer
void t()
{
    printf("TF  %d  %d\n", gi, &gi);
    gi = 7;
    return;
}

int main(void)
{
    int gi = 5;
    printf("TM  %d  %d\n", gi, &gi);
    t();
    printf("TM2 %d  %d\n", gi, &gi);
}

file: p2.c
PbV

- Finally, passing a variable
  - memory location of \(gi\) in \(t\) is different from in \(main\)
- Visible manifestation of PbV

```c
void t(int gi)
{
    printf("TF %d %d\n", gi, &gi);
    gi = 7;
    printf("TF2 %d\n", gi, &gi);
    return;
}

int main(void)
{
    int gi = 5;
    printf("TM %d %d\n", gi, &gi);
    t(gi);
    printf("TM2 %d\n", gi);
}
```
Return

• is also by value
• Must be else you would be getting a memory location from a stack frame that no longer exists

```c
file: p4.c

int t(int gi)
{
    printf("TF %d %d\n", gi, &gi);
    gi = 7;
    printf("TF2 %d %d\n", gi, &gi);
    return gi;
}

int main(void)
{
    int gi = 5;
    printf("TM %d %d\n", gi, &gi);
    int gii = t(gi);
    printf("TM2 %d %d\n", gii, &gii);
}
```
Pointer types

- `int *p;`
  - holds a pointer to an integer
  - this declaration is not pointing to anything
  - must point to a thing of the type
- All pointers are exactly the same size
  - Actually all pointers are exactly the same
  - So why the restriction that the pointer MUST point to something of it declared type?

Create a variable, `gi`, then create two variables that hold a pointer to `gi`.

VSC prefers first form

```c
int gi = 5;
int *pgi1 = &gi;
int * pgi2 = &gi;
int * pgi3 = &gi;
```
'*' Operator

* is also called the "indirection" operator

IMPORTANT
  * operator is not * in type declarations and is not multiply.
  • horrific

* operator works ONLY on pointer types
  * compile error
  * when you have a pointer
  • use * to mean "the value of the thing pointed to"
  • This is logic behind calling * an "indirection" operator

file: p5.c

```c
int main(void)
{
    int giv = 5;
    int *gip = &giv;
    printf("TM1%5d%12d%12d\n", giv, &giv, gip);
    *gip = 7; // set value into the pointer
    printf("TM2%5d%12d%12d%5d\n", giv, &giv, gip, *gip);
    // set value into the memory address
    //parens are required
    *(giv) = 9;
    printf("TM2%5d%12d%12d%5d\n", giv, &giv, gip, *gip);
}
```
Finally, PbR in C

- To get Pass by Reference in C
  - pass a pointer
  - use indirection operator to set the value into pointer

- Used this in HW1!
  - scanf

---

file: p6.c

```c
void t(int *gip) {
    printf("TT1%5d%12d\n", *gip, gip);
    *gip = 7;
    printf("TT1%5d%12d\n", *gip, gip);
}

int main(int argc, char const *argv[]) {
    int giv = 3;
    printf("TM1%5d%12d\n", giv, &giv);
    t(&giv);
    printf("TM2%5d%12d\n", giv, &giv);
    return 0;
}
```
Pointer and Casting

- Because all pointers are the same you can freely cast pointers to other types.
- Setting/reading — not so much
- Consider java
  - String s = new String(“A”);
    Integer i = (Integer)s;
    kind of legal to do but a bad idea

```c
int main(void)
{
    int iint = 5;
    int *intp = &iint;
    printf("T1int%12d%12d\n", iint, intp);
    *intp = 999999;
    printf("T2int%12d%12d\n", iint, intp);
    char *chrp = (char *)intp;
    *chrp = 'a';
    printf("T3chr%12c%12d\n", *chrp, chrp);
    printf("T3int%12d%12d\n", *intp, intp);
}
```
Pointers and arrays

- Arrays are already pointers!
  - So with array you are doing PbR

```c
file: p10.c

void parray(char id, int asz, int arr[asz]) {
    for (int i = 0; i < 10; i++)
        printf("%1c%3d%12d%12d%5d\n", id, i, arr, &arr[i], arr[i]);
}

int main(void)
{
    int a[10];
    for (int i = 0; i < 10; i++)
        a[i] = (i*29) % 17;
    char id = 'M';
    for (int i = 0; i < 10; i++)
        printf("%1c%3d%12d%12d%5d\n", id, i, a, &a[i], a[i]);
    parray('A', 10, a);
}
```
Lab — Regular Expressions

• Write regular expressions you could use in grep to find
  • Note that to actually use some of these REs with grep, use quotes
  • all lines with the character z
  • all lines with at least 2 instances of the character z
  • all lines with 2 z’s with at least one character between
    • so pizza would not match but pizzaz would
  • all lines that have at least 2 upper case vowels
  • all lines that have 2 upper case vowels (but not I) separated by 10 more more characters (an upper case vowel could be one of intervening characters.

• If you use /home/gtowell/Public/206/a4/dickens.txt for a test file then these are the number of lines that each grep should find
  • z: 3909, 2 z’s: 976, 2 z’s with a separator: 143, 2 UC vowels: 23877, 2 UC vowels, but not I, separated by at least 10 chars: 2967

• All I need is the 5 regular expressions, but showing the grep commands is OK also. Do NOT send complete results of each grep.