David Wells

Cs 240

12/10/2010

Assignment 5

**Source**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <malloc.h>

#include <math.h>

#define rotateleft(x,n) ((x<<n) | (x>>(32-n)))

#define rotateright(x,n) ((x>>n) | (x<<(32-n)))

int tripwire(char exe[100]);

void SHA1(unsigned char \* str1, unsigned long int \*h0, unsigned long int \*h1,unsigned long int \*h2,unsigned long int \*h3,unsigned long int \*h4, int binsize);

int readfile(char \*input, char alias[1000][100], char aliased\_cmd[1000][100], char history[1000][100], int \*HISTORYCOUNT, int \*histindex, char lastcmd[100]);

int shellcheck(char \*input, char \*PATH, int \*pathsize,char history[1000][100], char alias[1000][100], char aliased\_cmd[1000][100], int \*HISTORYCOUNT, int \*histindex, char lastcmd[100]);

int runcmd(char \*\*cmd, int pathspec);

int Parse(char \*\*cmd, char \*temp);

int pathcheck(char \*input, char \*PATH, int \*pathsize);

int aliascheck(char \*input, char alias[1000][100], char aliased\_cmd[1000][100]);

int addalias(char \*input, char alias[1000][100], char aliased\_cmd[1000][100]);

int removealias(char \*input, char alias[1000][100], char aliased\_cmd[1000][100], int aliasindex);

int traverse\_aliases(char \*input, char alias[1000][100], int searchsection, int aliastotal);

int stringcmp(char \*arr, char \*input, int stringsize, int stringlocation);

int syntaxcheck(char \*input, char operator);

int historyedit(char \*input, int \*HISTORYCOUNT); /\*takes input and compares to HISTORYCOUNT. will modify HISTORYCOUNT by reference\*/

int loghistory(char \*input, char history[1000][100], int HISTORYCOUNT);

int accesshistory(char \*input, char history[1000][100], int \*historyindex, char lastcmd[100]);

int grabnumber(char \*input, int index); /\*will convert indexed point in input (char) to a number (int)\*/

int main()

{

char \*cmd[100], input[100], exe[100], alias[1000][100], aliased\_cmd[1000][100], PATH[128], \*currpath, history[1000][100], lastcmd[100];

int m=0, pathspec=0, choice, shellcmd=0, execute\_success=0, pathsize=0, HISTORYCOUNT=100, histindex;

currpath=getenv("PATH"); /\*grabbing the true PATH\*/

while(currpath[pathsize]!='\0') /\*find end of PATH so it can append to it\*/

{

PATH[pathsize]=currpath[pathsize]; /\*copy pointer to chars to an array\*/

pathsize++;

}

PATH[pathsize]='\0'; /\*NULL end of string\*/

memset(alias, '\0', sizeof(char) \* 1000 \* 100);

memset(aliased\_cmd, '\0', sizeof(char) \*1000 \*100);

readfile(input, alias, aliased\_cmd, history, &HISTORYCOUNT, &histindex, lastcmd); /\*reads from rc files and executes settings\*/

while(1)

{

pathspec=readinput(input, lastcmd); /\*take input. pathspec is a flag that determines if the user made the command path specified\*/

shellcmd=shellcheck(input, PATH, &pathsize, history,alias, aliased\_cmd, &HISTORYCOUNT, &histindex, lastcmd); /\*is this just a shellcommand and needs no execution?\*/

if(shellcmd==0) /\*skips execution step if input was just a shell command\*/

{

if(pathspec!=1)

{

choice=tripwire(input);

}

else

{

choice=-1;

}

if(choice==1 || choice ==-1)

{

Parse(cmd, input); /\*parsing input into a 2 dimensional array so that execvp can use it\*/

execute\_success=runcmd(cmd, pathspec); /\*executes child process to run shell command\*/

if(execute\_success==1)

{

histindex=loghistory(input, history, HISTORYCOUNT); /\*if command was executed sucessfully then log it in the historyand update history size\*/

}

memset(input,'\0',sizeof(char)\*100);

}

}

else /\*if it was a successful shellcommand then log history\*/

{

histindex=loghistory(input, history, HISTORYCOUNT);

}

shellcmd=0;

execute\_success=0;

}

}

void SHA1(unsigned char \*str1, unsigned long int \*h0, unsigned long int \*h1,unsigned long int \*h2,unsigned long int \*h3,unsigned long int \*h4, int binsize) /\*SHA-1 code i found off the internet\*/

{

unsigned long int a,b,c,d,e,f,k,temp;

int j, m, i;

\*h0 = 0x67452301;

\*h1 = 0xEFCDAB89;

\*h2 = 0x98BADCFE;

\*h3 = 0x10325476;

\*h4 = 0xC3D2E1F0;

unsigned char \* str;

str = (unsigned char \*)(malloc(sizeof(unsigned char) \* binsize)+100);

strcpy((char \*)str,(const char \*)str1);

// int current\_length = strlen((const char \*)str);

int current\_length=binsize-100;

int original\_length = current\_length;

str[current\_length] = 0x80;

str[current\_length + 1] = '\0';

char ic = str[current\_length];

current\_length++;

int ib = current\_length % 64;

if(ib<56)

ib = 56-ib;

else

ib = 120 - ib;

for(i=0;i < ib;i++)

{

str[current\_length]=0x00;

current\_length++;

}

str[current\_length + 1]='\0';

for(i=0;i<6;i++)

{

str[current\_length]=0x0;

current\_length++;

}

str[current\_length] = (original\_length \* 8) / 0x100 ;

current\_length++;

str[current\_length] = (original\_length \* 8) % 0x100;

current\_length++;

str[current\_length+i]='\0';

int number\_of\_chunks = current\_length/64;

unsigned long int word[80];

for(i=0;i<number\_of\_chunks;i++)

{

for(j=0;j<16;j++)

{

word[j] = str[i\*64 + j\*4 + 0] \* 0x1000000 + str[i\*64 + j\*4 + 1] \* 0x10000 + str[i\*64 + j\*4 + 2] \* 0x100 + str[i\*64 + j\*4 + 3];

}

for(j=16;j<80;j++)

{

word[j] = rotateleft((word[j-3] ^ word[j-8] ^ word[j-14] ^ word[j-16]),1);

}

a = \*h0;

b = \*h1;

c = \*h2;

d = \*h3;

e = \*h4;

for(m=0;m<80;m++)

{

if(m<=19)

{

f = (b & c) | ((~b) & d);

k = 0x5A827999;

}

else if(m<=39)

{

f = b ^ c ^ d;

k = 0x6ED9EBA1;

}

else if(m<=59)

{

f = (b & c) | (b & d) | (c & d);

k = 0x8F1BBCDC;

}

else

{

f = b ^ c ^ d;

k = 0xCA62C1D6;

}

temp = (rotateleft(a,5) + f + e + k + word[m]) & 0xFFFFFFFF;

e = d;

d = c;

c = rotateleft(b,30);

b = a;

a = temp;

}

\*h0 = \*h0 + a;

\*h1 = \*h1 + b;

\*h2 = \*h2 + c;

\*h3 = \*h3 + d;

\*h4 = \*h4 + e;

}

}

int tripwire(char exe[100])

{

unsigned char \*binary, \*userbinary;

int i=0, stringsize, binsize;

unsigned long filelength;

unsigned long int a0,a1,a2,a3,a4,b0,b1,b2,b3,b4;

char rootpath[100]="/bin/";

char userpath[100]="./bin/whoa/"; /\*users path of executables\*/

char choice[100];

FILE \*rp,\*up;

strcat(rootpath, exe); /\*contatenates path with file so we can find it in the directory\*/

strcat(userpath, exe); /\* ditto \*/

stringsize=strlen(rootpath);

rootpath[stringsize-1]='\0'; /\*gets rid of newline\*/

stringsize=strlen(userpath);

userpath[stringsize-1]='\0'; /\*gets rid of new line\*/

rp=fopen(rootpath, "rb");

/\*if cant open file\*/

if(rp=='\0')

{

fprintf(stderr, "cant open file %s", rootpath);

}

else

{

fseek(rp, 0, SEEK\_END);

binsize=ftell(rp);

fseek(rp, 0, SEEK\_SET); /\*gets size of binary file\*/

binary=malloc(binsize);

memset(binary, '\0',binsize);

fread(binary, 1,binsize, rp);

fclose(rp);

SHA1(binary,&a0,&a1,&a2,&a3,&a4, binsize); /\*takes SHA1 of true binary file\*/

up=fopen(userpath, "rb");

fseek(up, 0, SEEK\_END);

binsize=ftell(up);

fseek(up, 0, SEEK\_SET);

userbinary=malloc(binsize);

memset(userbinary, '\0',binsize);

fread(userbinary,1, binsize, up);

fclose(up);

SHA1(userbinary,&b0,&b1,&b2,&b3,&b4, binsize); /\*takes SHA1 of user binary file\*/

}

if((a0==b0) && (a1==b1) && (a2==b2) && (a3==b3) && (a4==b4)) /\*are the two hashes the same?\*/

{

strcpy(exe,userpath);

return 1;

}

else

{

int yes=0;

printf("command %s has been altered\n", userpath);

printf("\ndo you wish to revert to a trusted version of this command?");

printf("\n%s"," yes/no>");

while(1)

{

do

{

choice[i]=getchar();

i++;

}while(choice[i-1]!='\n');

if(choice[0]=='y' && choice[1]=='e' && choice [2]=='s' && choice[3]=='\n')

{

strcpy(exe,rootpath);

return 1;

}

else if(choice[0]=='n' && choice[1]=='o')

{

return 0;

}

else

{

printf("%s", "please type yes or no\n\n");

}

}

}

free(binary);

free(userbinary);

}

int shellcheck(char \*input, char \*PATH, int \*pathsize, char history[1000][100], char alias[1000][100], char aliased\_cmd[1000][100], int \*HISTORYCOUNT, int \*histindex, char lastcmd[100])

{

int shellcmd=0;

shellcmd=pathcheck(input, PATH, pathsize); /\*is this a PATH modification or inquery\*/

if(shellcmd==1)

{

return 1;

}

shellcmd=aliascheck(input, alias, aliased\_cmd); /\*are they using aliases or add/removing them?\*/

if(shellcmd==1)

{

return 1;

}

shellcmd=historyedit(input, HISTORYCOUNT); /\*is this a HISTORYCOUNT modification or inquery?\*/

if(shellcmd==1)

{

return 1;

}

shellcmd=accesshistory(input, history, histindex, lastcmd); /\*is the user accessing a previous history command or substituting\*/

if(shellcmd==1)

{

return 1;

}

return 0;

}

int readinput(char \*input, char lastcmd[100])

{

int i=0, pathspec=0;

char userpath[100]="/net/classes/cs240/bin/";

i=0;

while(input[i]!='\0')

{

lastcmd[i]=input[i]; /\*stores command in last cmd reguardless of execution success so that "^" can be used\*/

i++;

}

lastcmd[i]='\0';

i=-1;

memset(input, '\0', sizeof(char)\* 100);

printf("\n$"); /\*command line symbol\*/

while(input[i]!='\n')

{

i++;

input[i]=getchar(); /\*grabing char by char from command line\*/

if(input[i]==EOF)

{

exit(0); /\*if user typed control D then terminate shell\*/

}

else if(input[i]=='/')

{

pathspec=1; /\*path specified input\*/

}

}

input[i+1]='\0'; /\*ending input with NULL for easier string parsing\*/

return pathspec;

}

int readfile(char \*input, char alias[1000][100], char aliased\_cmd[1000][100],char history[1000][100], int \*HISTORYCOUNT,int \*histindex, char lastcmd[100])

{

int shellcmd=0;

FILE \*twsrc=fopen(".twsrc","r"); /\*open twsrc file\*/

FILE \*twshistory=fopen(".tws\_history", "r"); /\*open history tws file\*/

if ((twsrc == '\0') || twshistory=='\0')

{

fprintf(stderr, "Can't open file twshistory or twsrc!\n");

exit(1);

}

while(fgets(input, 80, twsrc) != NULL) /\*read from twsrc file\*/

{

shellcmd=aliascheck(input, alias, aliased\_cmd);

}

while(fgets(input, 80, twshistory) != NULL) /\*read from twsrc file\*/

{

shellcmd=historyedit(input, HISTORYCOUNT);

shellcmd=accesshistory(input, history, histindex, lastcmd);

}

}

int runcmd(char \*\*cmd, int pathspec)

{

int status, success=0;

pid\_t pid=fork();

if(pid<0) /\* if it failed\*/

{

printf("could not make child process\n");

}

else if(pid==0 && pathspec==0) /\*child process\*/ /\*if not path specific command\*/

{

printf("no such command %d", wait(&status));

success=execvp(\*cmd, cmd);

exit(0);

}

else if(pid==0 && pathspec==1) /\*if path specific command\*/

{

success=execv(\*cmd, cmd);

pathspec=0;

}

else if(pid!=wait(&status)) /\*wait for child process to end\*/

{

printf("parent did not wait for child to end");

}

else

{

/\*do nothing\*/

}

if(success>=0)

{

return 1;

}

else

{

return 0;

}

}

int Parse(char \*\*cmd, char \*input)

{

int i=0, flag=0, col=0;

cmd[1]='\0'; /\*always setting cmd[1] to point to null\*/

while(input[i]!='\n')

{

if(input[i]=='\0')

{

break;

}

if((input[i]==' ' || input[i]=='\t') && flag==1) /\*NULLing spaces in input\*/

{

input[i]='\0';

flag=0;

}

else if(flag==0)

{

cmd[col]=&input[i];

flag=1;

col++;

}

i++;

}

input[i]='\0'; /\* nullifying the \n character from input\*/

}

int pathcheck(char \*input, char \*PATH, int \*pathsize)

{

int i=4, k=0, stringcmd=0, pathlength;

char pathstring[10]={'P', 'A', 'T', 'H'};

stringcmd=stringcmp(pathstring, input, 4,0);

if(stringcmd==1)

{

if(input[i]=='=') /\*is this an assignment of a new PATH?\*/

{

stringcmd=0; /\*clearing variable for future use\*/

while(input[i]!='\0')

{

if(input[i]=='=')

{

i++; /\*parse out equal sign\*/

stringcmd=stringcmp(pathstring, input, 4, i);

if(stringcmd==1) /\*if it equals "PATH" then give it the env var "PATH"\*/

{

while(PATH[k]!='\0') /\*find end of PATH so it can append to it\*/

{

k++;

}

i=i+4; /\*this allows for it to skip over the literal "PATH" in the input string after we already extracted what PATH means\*/

}

}

if(input[i]=='+')

{

PATH[k]=':';

i++;

k++;

}

if(input[i]==' ')

{

printf("%s", "please dont use spaces when specifying path");

return 1;

}

PATH[k]=input[i];

i++;

k++;

}

PATH[k]='\0';

printf("your path is now %s\n", PATH);

return 1;

}

else /\*if the user justs types "PATH" then simply display PATH\*/

{

printf("your path is now %s\n", PATH);

return 1;

}

}

else

{

return 0;

}

}

int aliascheck(char \*input, char alias[1000][100], char aliased\_cmd[1000][100])

{

int i, aliascmd=0, aliasindex, unaliascmd=0, q=0,m=0, p=0, correct\_syntax=0;

static int aliastotal;

char aliasstring[10]={'a', 'l', 'i','a', 's'}; /\*the string "alias" to compare to user input\*/

char unaliasstring[10]={'u', 'n', 'a', 'l', 'i', 'a', 's'}; /\*the string "unalias" to compare to user input\*/

aliascmd=stringcmp(aliasstring, input, 5, 0); /\*did the user type alias?\*/

unaliascmd=stringcmp(unaliasstring, input, 7, 0);

if(aliascmd==1) /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*if input is defining an alias\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

correct\_syntax=syntaxcheck(input, '='); /\*checks if correct syntax was used for alias\*/

if(correct\_syntax!=1) /\*if they used incorrect syntax, ignore command and restart input\*/

{

printf("\n%s\n", "Incorrect usage of alias. example: use alias ls=l");

}

else

{

aliastotal=addalias(input, alias, aliased\_cmd); /\*if they typed alias, then add the new alias\*/

}

return 1; /\*skip exec command\*/

}

else if(unaliascmd==1)

{

aliasindex=traverse\_aliases(input, alias, 8, aliastotal);

if(aliasindex==-1)

{

printf("%s\n", "no such alias defined");

return 1;

}

else

{

removealias(input, alias, aliased\_cmd, aliasindex);

return 1;

}

}

else /\*\*\*\*\*\*\*\*\*\*\*\*\*\*check if input is a pre-existing alias\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

m=traverse\_aliases(input,alias,0,aliastotal);

if(m!=-1) /\*if this is a pre defined alias then use it's real command\*/

{

while(aliased\_cmd[m][p]!='\0')

{

input[p]=aliased\_cmd[m][p];

p++;

}

input[p]='\0';

}

}

return 0;

}

int addalias(char \*input, char alias[1000][100], char aliased\_cmd[1000][100])

{

static int aliastotal=0;

int i=6,m=0; /\*i starts as 6 to skip over the "alias" command in string\*/

while(input[i]!='=')

{

alias[aliastotal][m]=input[i];

i++;

m++;

}

m=0;

i++; /\*skip '=' sign\*/

while(input[i]!='\0')

{

aliased\_cmd[aliastotal][m]=input[i]; /\*grab right side of equal sign and store real cmd in aliased\_cmd\*/

i++;

m++;

}

aliastotal++;

return aliastotal;

}

int removealias(char \*input, char alias[1000][100], char aliased\_cmd[1000][100],int aliasindex)

{

int p;

if(alias[aliasindex+1][0]!='\0')

{

while(alias[aliasindex+1][0]!='\0')

{

p=0;

while(alias[aliasindex+1][p]!='\0')

{

alias[aliasindex][p]=alias[aliasindex+1][p];

aliased\_cmd[aliasindex][p]=aliased\_cmd[aliasindex+1][p];

p++;

}

alias[aliasindex][p]='\0';

aliased\_cmd[aliasindex][p]='\0';

aliasindex++;

}

}

else

{

memset(alias[aliasindex], '\0', sizeof(char)\* 100);

memset(aliased\_cmd[aliasindex], '\0', sizeof(char) \* 100);

}

aliased\_cmd[aliasindex][0]='\0';

alias[aliasindex][0]='\0';

}

int traverse\_aliases(char \*input, char alias[1000][100], int searchsection, int aliastotal)

{

int n=0, p=0, k=0, i=searchsection;

while(alias[n][p]!='\0')

{

if(alias[n][p]==input[i]) /\*traverse aliases in search of portion of input specified by searchsection\*/

{

k=1;

p++;

i++;

}

else

{

k=0;

n++;

p=0;

i=searchsection;

}

}

if(k==1)

{

return n; /\*returns which block in the alias array that the alias is stored\*/

}

else

{

return -1;

}

}

int stringcmp(char \*arr, char \*input, int stringsize, int stringlocation)

{

int i=0, g=0, k=0;

for(i;i<stringsize;i++)

{

if(arr[k]==input[stringlocation])

{

g=1;

}

else

{

g=0;

return 0;

}

k++;

stringlocation++;

}

return 1;

}

int syntaxcheck(char \*input, char operator)

{

int correct\_syntax=0, q=0;

while(input[q]!='\0')

{

if(input[q]==operator)

{

correct\_syntax=1; /\*checks for correct usage of alias (i.e. alias ls=l)\*/

return 1;

}

q++;

}

if(correct\_syntax!=1) /\*if they used incorrect syntax, ignore command and restart input\*/

{

return 0;

}

}

int historyedit(char \*input, int \*HISTORYCOUNT)

{

int match=0, m=0, i=12,k=0; /\*i is 12 to skip "HISTORYCOUNT" in input\*/

char hist[16]={'H','I','S','T','O','R','Y','C','O','U','N','T'};

char numconvert[34];

match=stringcmp(hist, input, 12,0); /\*did the user type HISTORYCOUNT?\*/

if(match==1)

{

if(input[i]!='=') /\*if they are not assigning a new value to HISTORYCOUNT then just print out HISTORYCOUNT\*/

{

printf("HISTORYCOUNT is %d", \*HISTORYCOUNT);

return 1;

}

else /\*the user is re initializing HISTORYCOUNT\*/

{

i++;

k=grabnumber(input, i); /\*grabs number on right side of equalsign and stores in k for tests to be done on k\*/

if(k>1000)

{

printf("%s", "cannot have more than a 1000 history size");

\*HISTORYCOUNT=999;

}

else if(k==-1) /\*wrong data assignment to HISTORYCOUNT\*/

{

return 1;

}

\*HISTORYCOUNT=k; /\* k is worthy to be stored to HISTORYCOUNT\*/

}

return 0;

}

return 0;

}

int loghistory(char \*input, char history[1000][100], int HISTORYCOUNT)

{

static int histindex=0;

int i=0;

histindex++;

if(histindex<HISTORYCOUNT) /\*if current command has not exceeded historycount\*/

{

while(input[i]!='\0')

{

history[histindex][i]=input[i]; /\*store command in history\*/

i++;

}

history[histindex][i]='\0'; /\*ending history entry with NULL\*/

}

else

{

histindex=0;

memset(history,'\0',sizeof(char)\*1000\*100); /\*if you exceed HISTORYCOUNT buffer size, then clear old history\*/

while(input[i]!='\0')

{

history[histindex][i]=input[i]; /\*story command in history\*/

i++;

}

}

return histindex;

}

int accesshistory(char \*input, char history[1000][100], int \*histindex, char lastcmd[100])

{

int i=0,m=0,s=0, match=0, o,sub, subsize, sublocation=0, flag, badsection=0, newsection=0, gap;

char hist[16]={'h','i','s','t','o','r','y'};

match=stringcmp(hist, input,7,0);

if(match==1) /\*if user typed "history" then print out history\*/

{

while(i<=\*histindex-1)

{

printf("\n%d:%s\n", i+1, history[i+1]);

i++;

}

return 1;

}

if(input[0]=='!' && input[1]!='!')

{

i++;

s=grabnumber(input, i); /\*grabs which history command they want to execute\*/

s++; /\*incrimenting to compensate for user input to array\*/

if(s>\*histindex)

{

printf("%s", "cant access that far in history");

return 1;

}

while(history[s][m]!='\0')

{

input[m]=history[s][m];

m++;

}

return 0;

}

else if(input[0]=='!' && input[1]=='!') /\*if double bang, then excecute last command again\*/

{

while(history[\*histindex][m]!='\0')

{

input[m]=history[\*histindex][m]; /\*store last command in input\*/

m++;

}

input[m]='\n';

return 0;

}

else if(input[0]=='^')

{

m=0;

o=1;

match=0;

while(input[o]!='^' && input[o]!='\n' && lastcmd[m]!='\0')

{

if(lastcmd[m]==input[o])

{

match=1;

if(flag==0) /\*marks placement of string match\*/

{

sublocation=m;

flag=1;

}

o++;

m++;

}

else

{

flag=0;

m++;

o=1;

match=0;

}

}

if(match==1) /\*if it found a string to sub then perform sub\*/

{

badsection=o-1; /\*holding the start of index to be subbed\*/

while(input[o]!='\0' && input[o]!='\n')

{

o++;

}

newsection=o-badsection-2; /\*holds start of index that is the substitution\*/

o=badsection+1;

i=0;

if(input[o+1]=='\0') /\*if there was no second ^, then delete area (shift array over)\*/

{

i=sublocation;

gap=badsection-newsection-1;

while(lastcmd[i]!='\n')

{

lastcmd[i]=lastcmd[i+gap]; /\*shift array\*/

i++;

}

lastcmd[i]='\n';

lastcmd[i+1]='\0';

printf("%s", lastcmd);

}

else

{

o++;

sub=o;

i=sublocation;

if(newsection<badsection)

{

while(input[sub]!='\0' && input[sub]!='\n')

{

lastcmd[i]=input[sub]; /\*where the string is to be subbed, sub it with the part to the right of the second "^" in input\*/

sub++;

i++;

}

gap=badsection-newsection;

while(lastcmd[i+gap]!='\0')

{

lastcmd[i]=lastcmd[i+gap];

i++;

}

lastcmd[i]='\0';

}

else if(newsection==badsection)

{

while(input[sub]!='\0' && input[sub]!='\n')

{

lastcmd[i]=input[sub]; /\*where the string is to be subbed, sub it with the part to the right of the second "^" in input\*/

o++;

i++;

}

}

else if(newsection>badsection)

{

o=sub;

gap=newsection-badsection;

while(lastcmd[o]!='\n')

{

o++; /\*gives index to last element of lastcmd\*/

}

o--;

lastcmd[o+gap]='\0';

o--;

while((o+gap)!=sublocation)

{

lastcmd[o+gap]=lastcmd[o];

o--;

}

i=sublocation;

while(input[sub+1]!='\0')

{

lastcmd[i]=input[sub];

i++;

sub++;

}

}

i=0;

}

i=0;

while(lastcmd[i]!='\0') /\*perform substitution and store in input string\*/

{

input[i]=lastcmd[i];

i++;

}

input[i]='\n';

}

else /\*if no string was found to sub for\*/

{

return 1; /\*skip parse exec\*/

}

return 0; /\*continue to parse and exec\*/

}

}

int grabnumber(char \*input, int index)

{

char numconvert[16];

int m=0, num=0;

while(input[index]!='\n')

{

if(input[index]<'0' || input[index]>'9') /\*if input is not a number then cancel operation\*/

{

printf("%s", "cannot assign character to integer based variable");

return -1;

}

else

{

numconvert[m]=input[index]; /\*putting number to be assigned in a new string for type casting\*/

m++;

}

numconvert[m]='\0';

num=(atoi(numconvert) -1); /\*assigns new value to HISTORYCOUNT\*/

index++;

}

return num;

}

**TWS Manual**

**History:**

*Tws can recal previous commands, the syntax is as follows:*

**history-** prints out all commands you have typed for the log in session with a corresponding number

**!’number**’- will execute the corresponding command in your history list.

**!!**-will execute last command

**^**-can substitute part of string with new part. (example, if you typed: whokch then ^ok^i would sub “ok” with “i)

**HISTORYCOUNT**-env variable that display how far back your history will keep track of commands. (limit 1000)

**HISTORYCOUNT=’number’**- sets this env variable to new value.

**Aliasing**

*Tws can substitute commands with a string of your discretion***.**

**alias ‘string’=’command’**- from now on ‘string’ will execute ‘command’ when typed.

**unalias ‘string’**- this will erase previous alias

**Path**

*Tws can grab the PATH environment variable and append it.*

**PATH**- echo what PATH is.

**PATH=’new path’**- sets environment variable to ‘new path’ (also PATH=PATH+’new path’ will append to existing path)

**Other notes**

If the message “<command> has been altered do you wish to resort to a trusted version of this command yes/no>?” then this means that the excecutable in your user path has been altered and is not safe to run. By typing yes you are telling TWS to run the program from the root/bin, by typing no it skips running the command.