**CSE2035 C프로그래밍**

(설계 프로젝트 1 : Simple vi editor 제작)

담당교수: 서강대학교 컴퓨터공학과

이름:이세진 학번:20161619

1. **Design Problem & The Aim**

The aim of this project was to make a Simple Vi editor, which is 80x24 in size. The vi editor provides three different modes(command mode, editor mode, ex-command mode) in total. In the command mode, 0,$,h,j,k,l,i,o,dd,w,b were implemented and in the editor mode, users are able to enter a character, use a tab, enter, space and backspace. Lastly, in the ex-command mode, save,quit,save and quit and force quitting were implemented. My personal goal of this project was to learn how to apply the knowledge I’ve gained so far from this class to create something practical.

**2.1 Research**

Since this was my first time making a vi editor, I thought it was very essential to do some research on Vi editor beforehand. So I started off by learning about the different types of commands that exist in Vi editor and its functions. Fortunately, we’ve already been using puTTy, which is a type of vi editor, to code in class. Hence, I could simply test out each commands in puTTy and see how it operates. Such research helped me to get used to commands like ‘w’ or ‘/search’ which I’ve never used previously. Also, vi editor has to read a file and update the file so I needed to learn how to deal with files in C language using such functions like fscanf() and fopen().

**2.2. Analysis**

After gaining enough knowledge of Vi editor and understanding how to approach the problem, I began to design a list of functions needed to satisfy specific requirements. Since this was a relatively big project, I also needed to give a precise name for each variable and keep in track of how each function operates and what the return value is. To make this easier, I wrote a lot of comments to make my code understandable for other programmers.

I will explain what each variables do in the main function.

1. Char \* filename

- This array of char was used to store filename to be opened as soon as the program starts. If there is no such file with the same name, the program starts with a blank page(which means lines double array contains ‘\0’ values for all of its components.

1. File\* fin, File \*fout, File \*vim;

- There were three File\* variables handled during the program execution. First, fin is the file opened. Therefore, fin=fopen(filename,”r”), because we only need to read the contents of the file. On the other hand, fout is the file that we will later open to update the content. Hence, fout=fopen(filename,”w”), because we need to put the components of lines into the file. Lastly, we open another file in the program named, “.vimrc”. This is basically a file that contains optional settings to initialize Vim. For my vi editor, I’ve only implemented it to check if vimrc has set nu. If it has, it will print out the line numbers.

1. Int i,j,x,y,z,k

All of these integers weren’t given precise and specific names because I reused them when I used for loops, while loops and etc. However, when the program enters the first while loop, i and j almost always are used to represent the position of the cursor. i is the y position of the cursor while j is the x position of the cursor.

1. Int N,E,X

- In the main function, I’ve defined three int variables N, E and X. They stand for normal(command) mode, editor mode and ex command mode. When N is true, it meant the editor is in normal mode and if E is true, it meant it was in Edit mode and vice versa. By changing their values in the while loop, I was able to control the editor to be in the specific mode. This made sure the user isn’t allowed to enter a value in lines array when he is in ex-command mode or the user wouldn’t be able to save when in editor mode.

5.char c

- this variable C was constantly used whenever the editor received a char from a file or the user.

6.char \*entered,\*exco,\*co,\*order

* Char array was used to store commands received in ex-command mode and when a user inputs “/search” in command mode. Unlike other commands like ‘q’ or ‘w’ which only requires the user to put one character, commands in ex-command mode like :q! Or :set nu requires the user to put multiples of character.

Excluding the main function, I’ve made 7 functions in total.

1.void Space(int \*i, int \*j, char\*\* lines)

2.void backSpace(int \*i,int \*j, char\*\* lines)

3.void Enter(int \*i, int\*j, char\*\* lines)

4.void beforeWord(int \*i, int\*j, char\*\*lines)

5.void nextWord(int \*i,

int \*j, char\*\* lines)

6.int find(int \*i, int\*j, char\*\*lines, char\* exco, int length)

7.void save(char \*filename, char\*\* lines)

\*following functions will be explained in depth in the next section

It is important to notice that I wrote a separate function from the main function for two different cases. First, if the same algorithm is being used for more than one command. For example, when the user inputs a space, space function is called to update lines array and puts a space where the cursor had been. When the user inputs a tab, the exact same algorithm is required to update lines array. You simply need to call space function three times(or four times if the user has asked to) to implement tab. The other reason is if the algorithm is way too complicated to write in the main function. The function, nextWord was used to implement ‘w’ , which moves the cursor to the beginning of the next word. Since it had a lot of requirements and exceptions to consider, the algorithm itself was very long. Therefore, I’ve created a function to make it easier to read and edit(if required later).

**2.3 Making the program**

Here is the basic flowchart of the program. First, when the program is started, it finds a file with the name, filename which is stored in argv[1]. If it is found, it copies its contents to the double char array called lines. If such file does not exist in the user’s directory, lines will be filled with a null value, ‘\0’. Therefore, a blank page will be printed out when the user starts the editor. Next, it finds a file called ‘.vimrc’, which is a file containing optional settings to initialize Vim. For my vi editor, I have only implemented it in a way it can only recognize ‘set nu’ in .vimrc. If ‘set nu’ exists in the file, it will make sure the line numbers are printed out as soon as the editor gets printed out on the screen. Then, the program will enter a while loop. As you can see in the chart, this while loop consists of three other while loops, one for each mode. When the program begins, it enters the while loop for command mode. Hence, the editor is in command mode the moment it executes. In the while loop for command mode, it begins by calling draw function, which basically prints out the editor onto the user’s screen. Then, it receives a character from the user. The rest of while loop consists of implementations of command mode’s command like UP,DOWN,LEFT,RIGHT,h,j,k,l,o,dd,0,i,o,$,w,b,/seach and :. This while loop continues until the user’s input equals to “:” or “i” or “o”. If the user commands “:”, it escapes the command mode and enters the ex-command mode. On the other hand, it changes to editor mode when user inputs “i” or “o”. (how each commands function will be explained later). Similarly, every time the while loop of editor mode begins, it calls draw function to print out the editor. The editor mode is the only loop where the user can make actual changes to the lines array. Such functions like space, enter, backspace, tab, putting a character and ESC are implemented. Only when the user commands ‘ESC’ in the editor mode, the while loop of editor mode finishes and the program enters the while loop of command mode. In the ex-command while loop, more than one character is received from the user. In fact, it receives commands until the user input ENTER. Hence, an array is used to store the command. If the user’s command equals to “:set nu” or “:set expandtab” or “:#”(which puts cursor in that line number), the excommand while loop finishes and it enters the while loop for command mode. Meanwhile, if the user’s input equals to “q”, “q!” or “wq!”, the program is finally able to escape the outer while loop and hence the program finishes.

Explanation of the code

1. Initializing editor

int main(int argc,char \*argv[]) {

char \*filename = argv[1]; //argv[1] : edit filename

FILE \* fin,\*fout, \*vim; // variables were explained previously.

int i,j,x,y,z,k,N,E,X,check,h,flag,num,tabstop,vim\_found;

char c; //

char \*entered,\*exco,\*co,\*order;

char \*\*copied,\*\*vim\_info;

vim\_found=0;

check=0;

tabstop=3;

h=0;

flag=0;

co=(char\*)malloc(sizeof(char)\*20); // initializing co, entered, copied arrays through dynamic allocation;

entered=(char\*)malloc(sizeof(char)\*20);

copied=(char\*\*)malloc(sizeof(char\*)\*33);

vim\_info=(char\*\*)malloc(sizeof(char\*)\*33);

for(i=0;i<33;i++){

copied[i]=(char\*)malloc(sizeof(char)\*88);

vim\_info[i]=(char\*)malloc(sizeof(char)\*88);

for(j=0;j<88;j++)

{

copied[i][j]='\0'; // putting Null values to each array of copied and vim\_info. Hence, if the program cannot find .vimrc or

vim\_info[i][j]='\0'; // file input, the editor will start with a blank page.

}

}

for(x=0;x<20;x++)

{

entered[x]='\0';

co[x]='\0';

}

flag=0;

int mode=0; //mode 0=command 1=edit 2= quit

fin=fopen(filename,"r"); //opening files called filename and .vimrc

vim=fopen(".vimrc","r");

char \*\*lines = (char \*\*)malloc(sizeof(char\*)\*33);

for ( x = 0 ; x < 33 ; x++ ) {

lines[x] = (char\*)malloc(sizeof(char)\*88);

for ( y = 0 ; y < 88 ; y++ )

lines[x][y] = '\0';

}

if(vim!=NULL)

{

i=0;

j=0;

for(x=0;x<HEIGHT;x++)

{

for(y=0;y<WIDTH;y++)

{

c=fgetc(vim);

if(c!=EOF&&c!='\n'&&c!='\t'&&c!='\b'&&c!='\a'&&c!='\r') //gets information from .vimrc and copy the

{ //contents to vim\_info.

vim\_info[x][y]=c;

}else if(c==ENTER)

{

break;

}

}

}

order="set nu"; // normally, vimrc files contain more than one optional setting. However, my vi editor was designed so it

vim\_found=find(&i,&j,vim\_info,order,6); //only checks if there is “set nu”

if(vim\_found==1)

{

lines[HEIGHT][WIDTH]='1'; // Because lines[HEIGHT][WIDTH] is not controlled or altered in any other part of

} // the program, it will be always ‘\0’ until the user commands “set nu” in the

} // ex-command mode or the virmc file contains “set nu”

/\*

\* initializing editor start

\*/

if(fin!=NULL)//

{

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

{

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

{

c=fgetc(fin);

if(c!=EOF&&c!='\n'&&c!='\t'&&c!='\b'&&c!='\a'&&c!='\r')

{

lines[i][j]=c; // just like vim\_info, lines will get contents from file fin

copied[i][j]=c; // copied will copy the exact same info in order to check

}else if(c==ENTER) // if any updates have been made before quitting.

break;

}

}

}

/\*

\* initializing editor end

\*/

/\*

\* inf loop start

\*/

2. The Outer while loop

….

i=0; // initializing i and j to 0 so the cursor always starts at (0,0)

j=0;

N=true;

E=false;

X=false; // As mentioned before, N,E,X each stands for normal mode, edit mode and excommand mode

while ( true ) {

draw(lines,i,j); // draw editor, 0,0 : cursor position

printf("[NORMAL] > ./%s > [CURSOR]%d:%d\n",filename,i,j);

printf("--COMMAND--");

char c = getch(); // wait input character

….

.../\* implementation of three other while loops \*/

3. The while loop for command mode contains..

3.1 Implementation of UP and k

if(c==UP||c=='k') // if the char received in the command mode equals to UP or ‘k’

{

if(i!=0) // this condition makes sure we don’t face segmentation error and also the user won’t be

{ // able to move the cursor up when the cursor is at the very top of the editor.

for(x=WIDTH-1;x>=0;x--) // this loop finds the position of the last element in the previous

{ // line

if(lines[i-1][x]!='\0')

{

break;

}

}

if(x<j) //if the last element in the previous line is more far to the left than the current position of cursor, the cursor will

{ //move to the position of the last element. Otherwise, it will just not move in X direction but just move to the previous line.

j=x;

i--;

}else

{

i--;

}

}

3.2. Implementation of Down or j

else if(c==DOWN||c=='j') // if the char received in the command mode equals to DOWN or ‘j’

{

if(i<HEIGHT-1) // this condition prohibits the user from moving the cursor down when the cursor is already at the

{ // last line of the editor.

for(y=WIDTH-1;y>=0;y--) //this loop finds the last element in the next line.

{

if(lines[i+1][y]!='\0')

{

break;

}

}

if(y<0) // if y<0 at this moment, it means there is no element in the next line. Therefore, the cursor will not move down.

{

}

else if(y<j) // if the last element in the next line is more to the left than the current cursor position, the cursor will move to the last element in the nextline. Otherwise, the cursor will just move down by one line.

{

j=y;

i++;

}else

{

i++;

}

}

3.3. Implementation of Left or h

else if((c==LEFT)||(c=='h') // if the char received in the command mode equals to LEFT or ‘h’

{

if(j>0) // if the cursor is currently at the left farthest in the editor, the cursor won’t move to the left. Otherwise, it will move

{ //position to the left.

j--;

}

}

3.4. Implementation of RIGHT or l

else if((c==RIGHT)||(c=='l')) ) // if the char received in the command mode equals to RIGHT or ‘h’

{

if(j<WIDTH-1&&lines[i][j+1]!='\0')// this condition limits the cursor to only move to the right if the cursor isn’t at the right farthest and position right to the cursor isn’t empty.

{

j++;

}

3.5. Implementation of ‘i’ and ‘o’

...// }else if(c=='i')

{

E=true; // this makes the program to exit the while loop of editor mode and enter the while loop of normal mode

N=false; // which makes the editor to go from editor mode to command mode.

}else if(c=='o')

{

if(i<HEIGHT-1)// if the cursor is not at the last line, it will move all of next lines down by one line and will enter

{ j=0; // editor mode

i=i+1;

Enter(&i,&j,lines);

E=true;

N=false;

}

}

3.6. Implementation of ‘dd’

else if(c=='d')

{

c=getch();

if(c=='d') //since the program receives the value by getch(), it can only get one character. To know that the user

{ // has inserted ‘dd’, we need to get another char and check if it’s also ‘d’.

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

{

lines[i][y]='\0'; //this algorithm deletes the contents of the line the cursor is currently in

}

for(y=i;y<HEIGHT-1;y++) //this loop moves the contents in the next line of the cursor down by one line.

{

for(x=WIDTH-1;x>=0;x--)

{

lines[y][x]=lines[y+1][x];

}

}

for(y=0;y<WIDTH-1;y++)

lines[HEIGHT-1][y]='\0';// deleting the last line of the editor

for(y=0;y<WIDTH;y++)

{

if(lines[i][y]!='\0'&&lines[i][y]!=' ') // the cursor moves to the first element of the line it had previously

{ //been in.

j=y;

break;

}

}

}

3.7. Implementation of ‘0’,’^’ and ‘$’

else if(c=='0') // the cursor moves to the farthest left when ‘0’ is entered. Thus, the cursor moves to the 1st position (i,0)

{

j=0;

}else if(c=='^') // the cursor moves to the first character of the first

{ //word in the line

for(x=0;x<WIDTH;x++)

{

if(lines[i][x]!='\0'&&lines[i][x]!=' ') // finds the first element in that line of array that is not null or a space

{

j=x; //when found, move the cursor to that position.

break;

}

}

}else if(c=='$') // moves the cursor to the last element of the last word in that line

{

for(x=WIDTH-1;x>=0;x--)

{

if(lines[i][x]!='\0'&&lines[i][x]!=' ') // starting from the farthest right, checks if the element in that line of array is

{ // empty or space.

j=x; // if it’s empty, move the cursor

break;

}

}

3.8. Implementation of ‘w’ and ‘b’ and ‘:’

}else if(c=='w') // finds the word next to the cursor and move the cursor to the first element of that word.

{

nextWord(&i,&j,lines);

}else if(c=='b') // finds the previous word in the respect to the cursor’s position and moves the cursor to the first letter of

{ //that word.

beforeWord(&i,&j,lines);

}else if(c==':') // change from command mode to excommand mode

{

N=false; // since N is changed to false, the while loop for normal mode will finish. And the program will now enter

X=true; //excommand mode.

break;

}

3.9. Implementation of ‘/search’

else if(c=='/') //if the user input ‘/’ in the command mode, we know that the person wants to search a word.

{

for(x=0;x<20;x++)

co[x]='\0'; // initialize the co array will null values so every time this command is given, it won’t be affected by the last

printf("\n/"); // time it was called.

flag=1; // flag will signal the number of characters entered

co[0]='/'; // the first element of co will always be ‘/’

while(c!=ENTER) //the loop continues until the user input ENTER

{

c=getch(); //gets a char from the user

if(c==BACKSPACE&&flag==1) // if the user inputs BACKSPACE and the co array only contains ‘\, change from

{ //command mode to editor mode

N=false;

E=true;

break;

}else if(c==BACKSPACE) //if the user inputs BACKSPACE and the co array contains some elements, delete the last

{ // element entered and decrease flag by 1 (because there is now one less element in co)

co[flag]='\0';

flag--;

}else if(c==ENTER) // if the user input ENTER, the loop will immediately finish.

{

break;

}else

{ // if all of the previous conditions were not met, it means that the entered character should be

co[flag]=c; // stored in co array.

flag++;

}

printf("%c",c); //print the character received eachtime so the user can see what he has put in.

}

find(&i,&j,lines,co,flag-1); // calls find function to move the cursor to the first letter of the target word

X=false;

N=true; // change from excommand mode to command mode

break;

}

}

4. The while loop of editor mode

while(E==true)

{

draw(lines,i,j);

printf("[INSERT] > ./%s > [CURSOR]%d:%d\n",filename,i,j);

printf("--INSERT--");

c=getch();

…..

4.1 Implementation of ESC

if(c==ESC) // if the user inputs ESC, the editor changes from editor mode to command mode

{

N=true;

E=false;

}

4.2 Implementation of Space,backSpace,Enter

else if(c==' ') // when the user inputs space, the space function is called to make appropriate changes in the lines

{

Space(&i,&j,lines);

}

else if(c==BACKSPACE) //when the user inputs BACKSPACE, the backSpace function is called to make appropriate changes in

{ //the lines

backSpace(&i,&j,lines);

}

else if(c==ENTER) //when the user inputs ENTER, the enter function is called to make appropriate changes in lines

{

Enter(&i,&j,lines);

}

/\* those three functions are explained in detail later \*/

4.3 Implementation of Tab

else if(c==TAB) // if the user inputs TAB, the space function is called tabstop times. Tabstop equals to 3 in the beginning of the

{ //program. However, if the user commands “:set expandtab”, it will increase to 4.

if(!(i==HEIGHT-1&&j==WIDTH-4))

{

for(x=0;x<tabstop;x++)

Space(&i,&j,lines);

}

}

4.4 Implementation of Inserting a character

else

{

if(j!=WIDTH-1) // if the char given by the user had not met all the conditions previously, it means it is not a

{ // but a character to be stored in the lines array.

for(x=WIDTH-2;x>j;x--) // this loops moves the contents in the right of the cursor is in, one step to the right.

{

lines[i][x]=lines[i][x-1];

}

lines[i][j]=c; // puts the char in the lines array at the position of the cursor

j++; // moves the cursor to the right by one

draw(lines,i,j); // draws the editor

}

5. The while loop for ex-command

check=0; // check is used to show what to print below the editor in the excommand mode

while(X==true)

{

for(x=0;x<20;x++)

{

entered[x]='\0';

}

flag=0; // stores the number of characters received

draw(lines,i,j);

printf("[NORMAL] > ./%s > [CURSOR]%d:%d \n", filename,i,j);

switch(check)

{

case 1: printf("System> No write since last change(add ! to override)\n:"); break;

// this warning printed if the user commanded “:q” but the file has not been saved after the last change.

case 2: printf("System> saved \n:"); break; // excommand에서 :w 를 입력했을때 이 문구를 출력한다.

// this is printed after the user has commanded “:w”

default : printf(":"); break;

}

c=getch();

while(c!=ENTER) // gets a char from the user until he inputs ENTER

{

c=getch();

if(c==ENTER) // escapes the while loop as soon as ENTER is given by the user

break;

if(c==BACKSPACE&&flag==0) // if the user has put BACKSPACE and the number of characters stored in

{ // entered equals to 0, the editor changes from excommand mode to command.

X=false; // mode

N=true;

break;

}else if(c==BACKSPACE) // if the user has put BACKSPACE and the number of characters stored in entered is not 0,

{ // erase the last character stored and update the number of characters in entered by subtracting

entered[flag]=' '; // it by 1.

flag--;

}else

{

entered[flag]=c; // if the user has put anything other than BACKSPACE, it is stored in entered array.

Flag++; // update the number of characters stored in entered

}

printf("%c",c); //printing out the character received

}

exco=(char\*)malloc(sizeof(char)\*flag); //dynamic allocation to create exco array

for(x=0;x<flag;x++)

{

exco[x]=entered[x]; //copy the contents of entered to exco.

}

}

}

5.1 Implementation of ‘:w’

if(exco[0]=='w'&&flag==1)// when the first element of the exco equals to ‘w’ and it is the only character stored, it means it is

{ // ‘w’ command.

save(filename,lines); // call save function to save the contents of lines into the file

for(x=0;x<HEIGHT;x++)

for(y=0;y<WIDTH;y++)

{

copied[x][y]=lines[x][y]; // update the value of copied array

}

check=2; // this makes the editor to print out “system>saved” in the excommand mode.

}

5.2 Implementation of ‘:wq’

else if(exco[0]=='w'&&exco[1]=='q') //if the ‘wq’ is commanded, call save function and quit the program.

{

save(filename,lines);

printf("\n");

return 0;

}

5.3 Implementation of ‘:q’

else if(exco[0]=='q'&&flag==1) .

{ h=0;

check=0;

for(x=0;x<HEIGHT;x++)

for(y=0;y<WIDTH;y++)

{

if(copied[x][y]!=lines[x][y]) // check if there is any different between copied array and lines.

{

check=1; // if different, increment h and update check to 1

h++;

break;

}

}

if(h==0) // if h is 0 it means all the contents of copied array and lines are the same. Therefore, the program can finish.

{

printf("\n");

return 0;

}

}

5.4 Implementation of ‘:q!’

else if(exco[0]=='q'&&exco[1]=='!') // Force quit by returning 0.

{

printf("\n");

return 0;

}

5.5 Implementation of ‘:set nu’ and ‘set nonu’

else if(strcmp(exco,"set nu")==0) // if the command equals to ‘set nu’, make lines[HEIGHT][WIDTH]=’1’. This gives a signal in draw

{ // function to print out number line

lines[HEIGHT][WIDTH]='1';

X=false;

E=true;

break;

}else if(strcmp(exco,"set nonu")==0) // by setting lines[HEIGHT][WIDTH] back to null value, the draw function no longer

{ // prints out number line

lines[HEIGHT][WIDTH]='\0';

X=false;

E=true;

break;

}

5.6 Implementation of ‘:set expandtab’

else if((strcmp(exco,"set expandtab")==0)||(strcmp(exco,"set tabstop=4")==0))

{ // make tabstop to 4. And change from excommand mode to command mode

tabstop=4;

X=false;

N=true;

break;

}

5.7 Implementation of ‘:#’

else if(exco[0]>='0'&&exco[0]<='9'&&(exco[1]>='0'&&exco[1]<='9'||exco[1]=='\0'))// this condition is met only when the user inputs a

{ // number

if(exco[1]=='\0') // changing exco[0] into int type if the user has only given one number

{

num=(int)(exco[0]-'0');

}

else

num=((int)exco[0]-'0')\*10+(int)(exco[1]-'0'); // converting exco[0] and exco[1] into the appropriate number.

for(z=HEIGHT-1;z>=0;z--)

{

for(k=WIDTH-1;k>=0;k--)

{

if(lines[z][k]!='\0') // finds the position of the last element in lines

{

x=z;

z=0;

}

}

}

if(num>x||(exco[2]>='0'&&exco[2]<='9') )// if the given number is greater than the last line in lines or the user has

{ // input more than 2 digits, the cursor moves to the first position in the last line.

i=x-1;

j=0;

}else if(num<0) // if the given line number is less than 0, move the cursor to( 0,0)

{

i=0;

j=0;

}else

{ //Otherwise, move the cursor to the first position in the desired line number

i=num-1;

j=0;

}

X=false; //change from excommand mode to command mode

N=true;

}

6. Draw function

void draw(char \*\*lines,int y,int x) // I have changed draw function slightly so it prints out the line number if

{ // lines[HEIGHT][WIDTH]==’1’.

int i,j;

if(lines[HEIGHT][WIDTH]!='1') // default

{ system("clear");

printf("┏");

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

printf("━");

printf("┓\n");

for ( i = 0 ; i < HEIGHT ; i++ ) {

printf("┃");

for ( j = 0 ; j < WIDTH ; j++ ) {

if ( y == i && x == j ) {

printf("I");

} else if(lines[i][j]=='\0')

printf(" ");

else {

// if(lines[i][j]!=ENTER&&lines[i][j]!='\n')

printf("%c",lines[i][j]);

}

}

printf("┃");

puts("");

}

printf("┗");

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

printf("━");

printf("┛\n");

}else

{ //this is when the user has commanded “:set nu” or the vimrc file contains set nu.

system("clear");

printf("┏");

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

printf("━");

printf("┓\n");

for ( i = 0 ; i < HEIGHT ; i++ ) {

if(lines[i][0]!='\0') // if there is an element in this line, print out the line number

{if(i<9)

{ printf("| %d ",i+1);

}else

printf("|%d ",i+1);

}

else printf("| "); // if there is no element in that line, only print out the left side of the box.

for ( j = 0 ; j <WIDTH-3 ; j++ ) { // since we need a space to print the line number, decrease the number of elements to be

if ( y == i && x == j ) { // printed in each line.

printf("I");

} else if(lines[i][j]=='\0')

printf(" ");

else {

printf("%c",lines[i][j]);

}

}

printf("┃");

puts("");

}

printf("┗");

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

printf("━");

printf("┛\n");

}

}

7. Space function

void Space(int\* i,int \*j, char\*\* lines)

{

int x;

if(lines[\*i][WIDTH-1]!='\0') // if there is an element in the farthest right in the line, the function finishes.

return;

{

for(x=WIDTH-1;x>\*j;x--) // this for loop moves the characters in the line to the right

{

lines[\*i][x]=lines[\*i][x-1];

}

lines[\*i][\*j]=' '; // put ‘ ‘ into the array empty where the cursor had been.

if(\*j<WIDTH-1) // if the cursor is not located in the farthest right, move the cursor to the right.

\*j=\*j+1;

else if(\*i<HEIGHT-1) // if the cursor had been in the farthest right, move the cursor

{ // in the first position in the next line

\*i=\*i+1;

\*j=0;

}

}

}

7.Backspace function

void backSpace(int\*ip,int\*jp, char\*\*lines)

{

int i,j,x,y,k,flag;

i=\*ip;

j=\*jp;

if(j==0&&i!=0) // if the cursor is at the first position but not the first line

{

for(x=WIDTH-1;x>=0;x--) // finds the last element in the previous line

{

if(lines[i-1][x]!='\0')

{

break;

}

}

flag=x; // flag shows how many characters can be inserted in the previous line

for(y=0;y<=WIDTH-1;y++)

{

if(lines[i][y]=='\0') // when this loop stops, y will be the number of characters in the line where the cursor is in.

{

break;

}

}

for(k=0;k<=y-1;k++) // In the previous line, move the characters that were in the right side of the cursor

{

lines[i-1][x+1+k]=lines[i][k];

}

for(x=i;x<HEIGHT-1;x++) // move down all the lines after the cursor by one line {

for(y=0;y<WIDTH;y++)

{

lines[x][y]=lines[x+1][y];

}

}

for(y=0;y<WIDTH;y++) //delete the last line

{

lines[HEIGHT-1][y]='\0';

}

\*ip=\*ip-1;.

\*jp=flag+1;

}

else if(!(i==0&&j==0)) //Otherwise, if the cursor is not located in the first position, just move all elements to the right

{

for(x=j-1;x<=WIDTH-1;x++)

{

lines[i][x]=lines[i][x+1];

}

lines[i][WIDTH-1]='\0'; //delete the last element in that line

if(j>0) //if the cursor is not in the farthest right, move the cursor to the left.

{

\*jp=\*jp-1;

}

}

}

8. Enter function

void Enter(int \*i, int\*j,char\*\*lines)

{

int x,y,z,k,flag;

z=\*j;

k=\*i;

for(x=0;x<WIDTH;x++) // if there is any element in the HEIGHT-1 th line, finish the function

{

if(lines[HEIGHT-1][x]!='\0')

{

return;

}

}

if(\*i!=HEIGHT-1)

{

for(x=HEIGHT-1;x>\*i;x--) // Move down all the lines that is two lines below the cursor

{

for(y=0;y<=WIDTH-1;y++)

{

lines[x][y]=lines[x-1][y];

}

}

for(y=0;y<=WIDTH-1;y++) //delete the nextline of the cursor

{

lines[\*i+1][y]='\0';

}

for(y=0;y<=\*j;y++) // Move the contents right to the cursor to the next line

{

lines[\*i+1][y]=lines[\*i][\*j];

\*j=\*j+1;

if(\*j==WIDTH-1)

{

\*j=0;

\*i=\*i+1;

break;

}

}

for(x=WIDTH-1;x>=z;x--) // delete the contents that was in the right side of the cursor in the line

lines[k][x]='\0';

if(z==0) // if the cursor is not in the farthest left, put ‘ ‘ in that position.

{

lines[k][0]=' ';

}

}

}

9. Find function

int find(int\*i,int\*j,char\*\*lines,char\*exco,int length) // this function is used in ‘/search’ command and also to find set up in vimrc file

{

int x,y,k,check,found;

found=-1;

for(x=\*i;x<HEIGHT;x++) // start searching from the right of the cursor

{

k=1; // each time the word is not found in a line, set k = 1 check = 0.

check=0; // k is 1 because in the exco[0] there is always ‘/’.

for(y=0;y<WIDTH;y++)

{

if(x==\*i&&y<=\*j) //this condition allows the program to not search anything in the left of the cursor in i th line

{

}

else if(lines[x][y]==exco[k]) // this is when the element of the exco matches with an element with lines.

{

k++; // add k by 1 so now it searches for the second letter of desired word

check++; // add check by 1. This keeps in track of the number of letters found

}else

{

k=1; // if the element of lines does not match the element of exco, set k and check back to 1 and 0.

check=0;

}

if(check==length) // when check equals to length, it means it has found the word.

{

\*i=x; // set the cursor position to the beginning of the word

\*j=y-length+1;

y=WIDTH;

x=HEIGHT;

found= 1; // found 1 means the word is found. -1 means it is not found

}

}

}

return found;

}

10.save function

void save(char\* filename, char\*\*lines) // storing elements in lines to a file with the name, \*filename.

{

FILE \*fout;

int x,y;

fout=fopen(filename,"w");

for(x=0;x<HEIGHT;x++)

{

for(y=0;y<WIDTH;y++)

{

if(lines[x][y]!='\0') // if lines[x][y] is not empty, put it to the file

fputc(lines[x][y],fout);

}

fputc(ENTER,fout);

}

fclose(fout);

}

10.nextWord function

void nextWord(int\* i, int\*j,char\*\* lines) // finds the first word next to the cursor and locates the cursor to the first element of the word

{

int x,y;

for(y=\*i;y<HEIGHT;y++)

{

for(x=0;x<WIDTH-1;x++)

{

if(y==\*i&&x<=\*j) // ignore the informations that are in the left of the cursor

{

}

else if(x==0&&lines[y][x]!=' '&&lines[y][x+1]!='\0') // if the character is in the farthest left and the next { // element is not empty, move the cursor to the farthest left.

\*i=y;

\*j=x;

y=HEIGHT;

x=WIDTH;

break;

}

else if(lines[y][x]=='\_') // if the character is ‘\_’ {

if(((lines[y][x-1]>='a'&&lines[y][x-1]<='Z')||(lines[y][x-1]>='A'&&lines[y][x-1]<='Z')||(lines[y][x-1]>='0'&&lines[y][x-1]<='9')))

// move the cursor to where the ‘\_’ is if there is no letter or alphabet infront of ‘\_’

{

\*i=y;

\*j=x;

y=HEIGHT;

x=WIDTH;

break;

}

}

else if((lines[y][x]>=33&&lines[y][x]<=47)||(lines[y][x]>=58&&lines[y][x]<=64)||(lines[y][x]>=91&&lines[y][x]<=96)||(lines[y][x]>=123&&lines[y][x]<=126))

{ //if the character is a punctuation mark

if(!((lines[y][x-1]>=33&&lines[y][x-1]<=47)||(lines[y][x-1]>=58&&lines[y][x-1]<=64)||(lines[y][x-1]>=91&&lines[y][x-1]<=96)||(lines[y][x-1]>=123&&lines[y][x-1]<=126))) // move the cursor to where the punctuation mark is if there is no punctuation mark right before it

{

\*i=y;

\*j=x;

y=HEIGHT;

x=WIDTH;

break;

}

}else if((lines[y][x]>='a'&&lines[y][x]<='z')||(lines[y][x]>='A'&&lines[y][x]<='Z')||(lines[y][x]>='0'&&lines[y][x]<='9'))

{ // if the character is an alphabet or a number

if((lines[y][x-1]==' ')||(lines[y][x-1]>=33&&lines[y][x-1]<=47)||(lines[y][x-1]>=58&&lines[y][x-1]<=64)||(lines[y][x-1]>=91&&lines[y][x-1]<=96)||(lines[y][x-1]>=123&&lines[y][x-1]<=126))// put the cursor to where the character is only if there is a space or a punctuation mark in front.

{

if(!(lines[y][x-1]=='\_')) // put the cursor to where the character is only if ‘\_’ isn’t in front of it.

{

\*i=y;

\*j=x;

y=HEIGHT;

x=WIDTH;

break;

}

}

}

}

}

}

12. beforeWord function

void beforeWord(int\*i,int\*j,char\*\*lines) // moves the cursor to the first element of the word that is in the left side of the cursor

{

int x,y;

for(y=\*i;y>=0;y--)

{

for(x=WIDTH-2;x>=0;x--)

{

if(y==\*i&&x>=\*j) // ignore any information that is in the same line as the cursor and also in the right side of the cursor.

{

}

else if(x==0&&lines[y][x]!=' '&&lines[y][x+1]!='\0'&&lines[y][x+1]!=' ') // if the character is in the farthest left and the 2nd element is also not empty, move the cursor to the farthest left.

{

\*i=y;

\*j=x;

y=-1;

x=-1;

break;

}

else if(lines[y][x]=='\_') //if the character is ‘\_’, move the cursor only if there isn’t an alphabet or letter in the left side.

{

if((lines[y][x-1]>='a'&&lines[y][x-1]<='Z')||(lines[y][x-1]>='A'&&lines[y][x-1]<='Z')||(lines[y][x-1]>='0'&&lines[y][x-1]<='9'))

{

\*i=y;

\*j=x;

y=-1;

x=-1;

break;

}

}else if((lines[y][x]>=33&&lines[y][x]<=47)||(lines[y][x]>=58&&lines[y][x]<=64)||(lines[y][x]>=91&&lines[y][x]<=96)||(lines[y][x]>=123&&lines[y][x]<=126))

{ //if the character is a punctuation mark, move the cursor to that position only if the next or previous character is a space or a letter or an alphabet.

if((lines[y][x-1]==' ')||(lines[y][x-1]>='a'&&lines[y][x-1]<='z')||(lines[y][x-1]>='A'&&lines[y][x-1]<='Z')||(lines[y][x-1]>='0'&&lines[y][x-1]<='9')||(lines[y][x+1]>='a'&&lines[y][x+1]<='z')||(lines[y][x+1]>='A'&&lines[y][x+1]<='Z')||(lines[y][x+1]>='0'&&lines[y][x+1]<='9'))

{

\*i=y;

\*j=x;

y=-1;

x=-1;

break;

}

}else if((lines[y][x]>='a'&&lines[y][x]<='z')||(lines[y][x]>='A'&&lines[y][x]<='Z')||(lines[y][x]>='0'&&lines[y][x]<='9'))

//if the letter is an alphabet or a number, put the cursor to that position only if there is a space or a punctuation mark that is not ‘\_’ before the letter

{

if((lines[y][x-1]==' ')||(lines[y][x-1]>=33&&lines[y][x-1]<=47)||(lines[y][x-1]>=58&&lines[y][x-1]<=64)||(lines[y][x-1]>=91&&lines[y][x-1]<=96)||(lines[y][x-1]>=123&&lines[y][x-1]<=126))

{

if(!(lines[y][x-1]=='\_'))

{

\*i=y;

\*j=x;

y=-1;

x=-1;

break;

}

}

else if(!((lines[y][x-1]>='a'&&lines[y][x-1]<='z')||(lines[y][x-1]>='A'&&lines[y][x-1]<='Z')||(lines[y][x-1]>='0'&&lines[y][x-1]<='9')))

{

\*i=y;

\*j=x;

y=-1;

x=-1;

break;

}

}

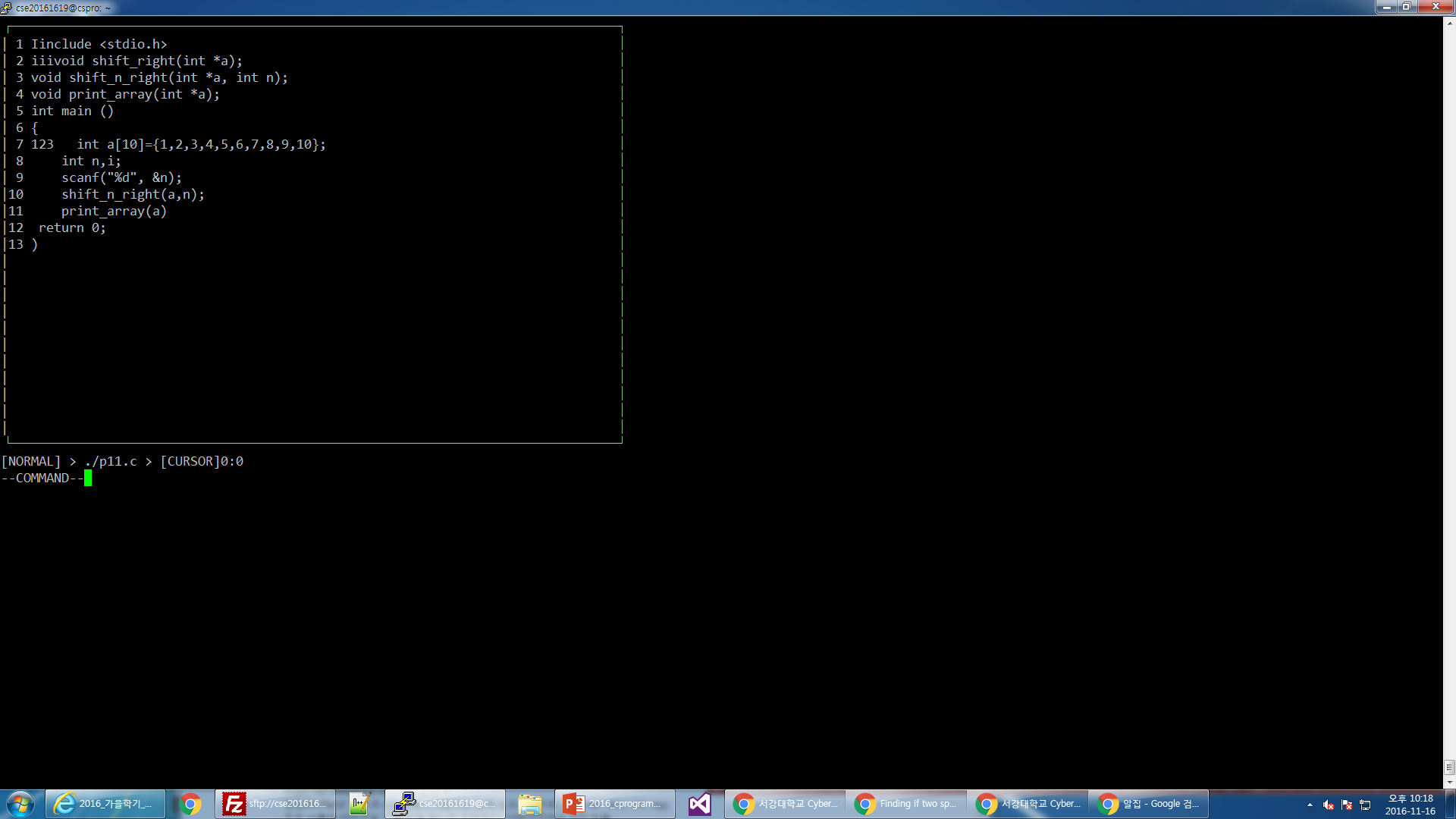
}

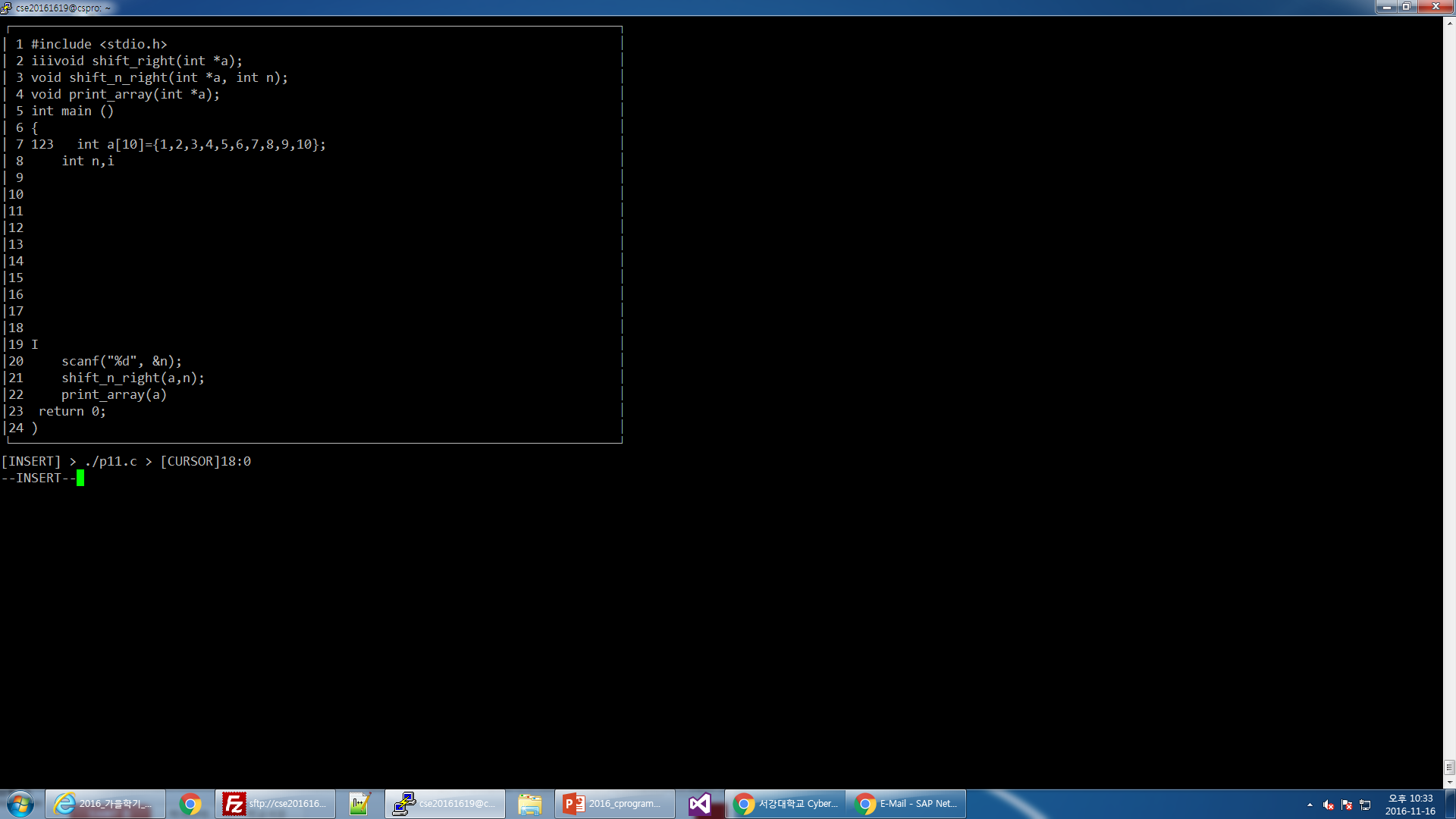
}

}

2.4. Test Case

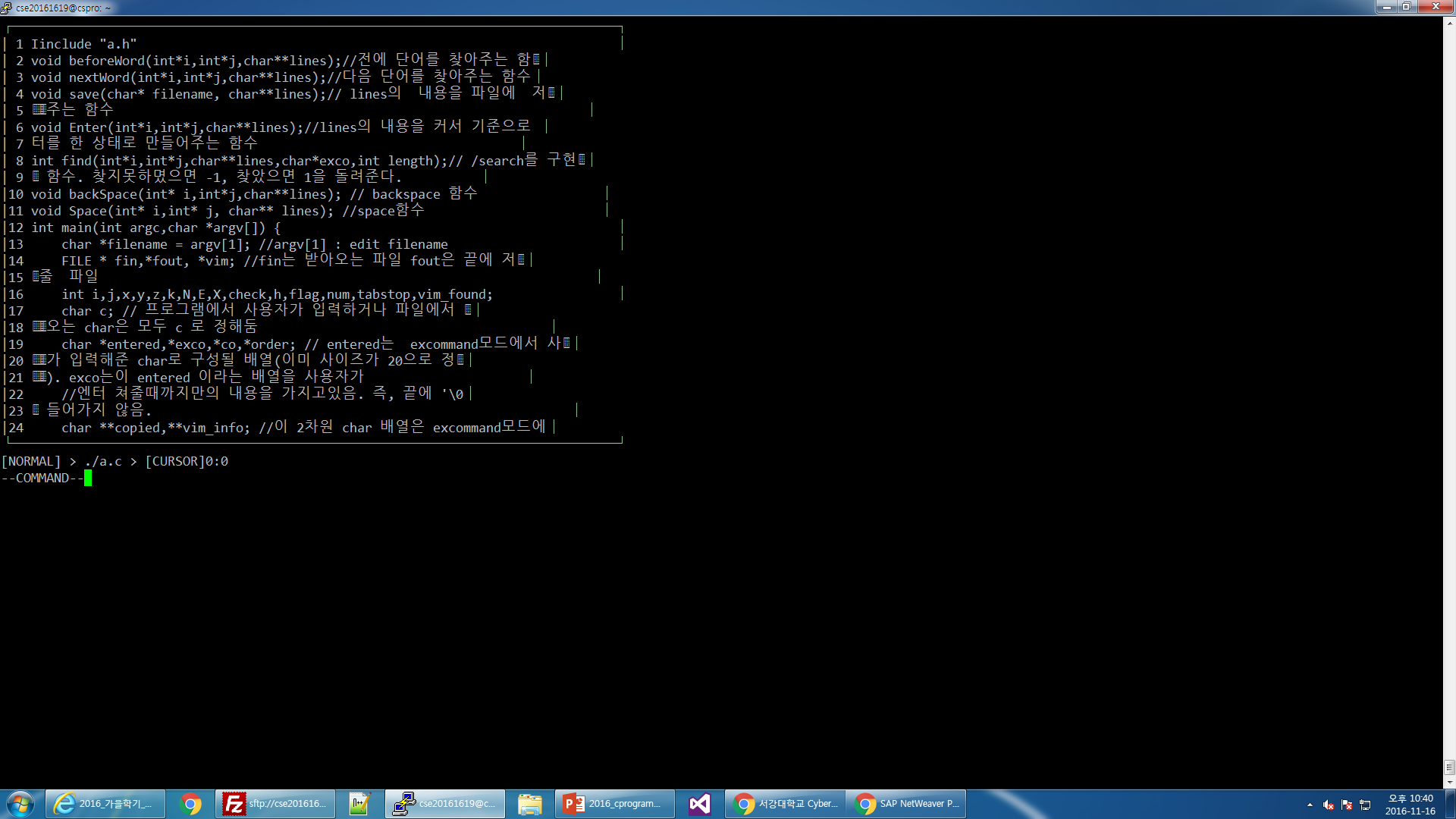
After I was done with writing the code, I ran the program with many different test cases to make sure there aren’t any fatal errors in vi editor.





For example, this is a case to test the function Enter. I kept putting ENTER to the editor until the last element in lines reached 24th line. The first time I’ve done the test, I did not write the condition where ENTER doesn’t work if there is an element in the last line. So when I entered in that situation, the last line was being deleted. Through such tests, I realized mistakes in my algorithm.

2.5 Evaluation



Through such test cases, I realized the shape of my box and a lot of functions implemented gets destroyed when the file fills in lines completely. However, I was told that we do not have to think about the cases where the line is completely filled. I have written all of my codes thinking that each lines[i] ends with ‘\0’, meaning that lines[i][WIDTH--1]=’\0’. Even though the program did not work properly if this was not the case, I still did not made any changes because I met all the requirements listed and was told to not think about such cases. If I had more time to work on this project, I will definitely change some part of the program so my vi editor can function properly in all test cases.

2.6 Stability of the program

Since we were prohibited from using static array and global variable, I used dynamic array and pointers throughout the program. Even though they are very practical and useful, I felt like they could easily cause serious problems. Trying to access an array outside its range will give segmentation error. Therefore, throughout the program, I was very careful that it never tries to access any array outside its range. Also, since we used a dynamic allocation, I used free() function at the end of the program to increase the stability of the program.

2.7 Productivity & durability

I think I had used my algorithm very productively, because I made functions like space() because such algorithm is also required to implement tab. However, I think my code was way too repetitive and long. Because of its length, I had trouble keeping in track of my own algorithm. If I have another chance to do this project, I will try to make more functions to make the program more modularized and easy to follow for other programmers.

1. **Extra**
2. 3.1 Generating environment

We were given a desktop with ssh connecting program to access a Linux server, where we had enough memory space to develop the vi editor.

3.2 Note

Upon completing this project, I feel really accomplished and proud of my work. This project allowed me to get used to using the knowledge and skills learned in class to make something very practical like Vi editor. I also learned that it is equally important and crucial to test the program with different test cases. Such practice allowed me to avoid having fatal errors in my program.

3.3 Making a team

An individual makes one team for this project.

3.4 Due date

2016/11/16 (Wednesday) 23:59:59

I was given 2 weeks to generate a vi editor and write documents to show how I approached the problem.