# C Programming
# A Simple Text-based Editor

---

Simted3

**What is Simted3?**

Simted3 stands for Simple Text-based Editor. It is a command-shell based text editor that enable user to edit, save, and view any type of file in ASCII format. The original purpose of this program is to demonstrate pointers, memory allocations/de-allocations, file input or output, and string search techniques in C.

**The Structure of the Program**

A huge buffer is used to read from and store all the text. The buffer is written in a linked-list structure, the text file is break down and represented in list of nodes. A second buffer is also used for tracking purposes for the undo feature in this illustration. Adding, replacing, deleting a line or a block involves simple iteration processes that involve O(n) complexity. Saving, reading, quitting, and undo are also bounded in the same complexity.

**How to use the program?**

After compilation, an executable would be automatically generated (default executable file name for C Programming Language is a.exe). Run the program with desired text file name in shell command line. Usage, "./simted3 [-c] foo.txt" .

**Limitations of the program**

The text program only reads in ASCII format text files. Any other file would not be able to work properly with Simted3.

**Other aspect of the program**

Undo is a complex operation that involves many different programming techniques in this demonstration. The operation would require a history of the command and the information the user had type. Using this information, the software tracks the amount of code that it needs to bring back or erase.

One other aspect of the program is to free the allocated memory when it is no longer to be used. In Java, this is automatically done with the Java's Garbage Collector. However, in C we will need to use Malloc () to reserve a memory space for our linked-list and use Free() when discarding the unused memories.

Currently, Simted3 supports add, delete, replace, add block, undo, print, save, and quit.

## Usage and Agreement

The author of the program holds no responsibilities towards the consequences this demonstration might incur to user's property. By using this program, the user has agreed to explore this demonstration at his or her own risks.

This software is open sourced. Anyone has the authorization to own, duplicate, or alter the source at his or her interest. The author only asks that his credentials remain intact.

## Contact Information

Original Author: Jason T Wu (jtywu at aim.com)

```c
/* Original Author: Jason T Wu */
/* ****Simple Editor written in C using Linked List with undo**** */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_CHAR 81
#define MAX_FN_CHAR 10
#define MAX_CMDCHAR 5

/* ****Classes**** */
typedef struct node{
	char content[MAX_CHAR];
	int val;
	void *next;
}node;

/* ****representation of the whole array list**** */
typedef struct list{
	node *HEAD;
	node *TAIL;
	int lineNum;
}whole_list;

/* ****Reading and adding lines to our buffer**** */
void read(whole_list *fileBuffer, char *readedLine){
	/* initializing a new node for appending*/
	node *newNode;
	newNode = (node *)malloc(sizeof(node));
	strcpy(newNode->content, readedLine);
	newNode->next = NULL;

	/* operation on appending new nodes */
	if(fileBuffer->HEAD == NULL){
		/* fileBuffer list is empty */
		fileBuffer->HEAD = newNode;
		fileBuffer->TAIL = newNode;
	}
	else {
		fileBuffer->TAIL->next = newNode;
		fileBuffer->TAIL = fileBuffer->TAIL->next;
	}
}
void addInt(whole_list *fileBuffer,int atLine,int myVal){

	node *i;
	int counter;
	counter = 1;

	/* Creating a new node for adding into our list */
	node *newNode;
	newNode = (node *)malloc(sizeof(node));
	newNode->val=myVal;
	newNode->next = NULL;


	if(atLine==0&&fileBuffer->HEAD==NULL&&fileBuffer->TAIL==NULL){
		newNode->next = fileBuffer->HEAD;
		fileBuffer->HEAD = newNode;
		fileBuffer->TAIL = newNode;
	}
	else if(atLine==0){

		newNode->next = fileBuffer->HEAD;
		fileBuffer->HEAD = newNode;
	}
	else{
		for(i=fileBuffer->HEAD;i !=NULL; i=i->next){
			if(counter==atLine){
				/* add */
				newNode->next = i->next;
```

```
                                i->next = newNode;
                        }
                        counter++;
                }
                                if(atLine==fileBuffer->lineNum){
                                        /* update tail if add at last line */
                                        fileBuffer->TAIL=newNode;
                                }
        }
        fileBuffer->lineNum=fileBuffer->lineNum+1;
}

/* ****free functions**** */
void freeList(whole_list *yourList){
        /* free all nodes in a list */
        node *tmpNode;
        node *i;
        for(i=yourList->HEAD;i!=NULL;i=i->next){
                tmpNode = yourList->HEAD->next;
                free(yourList->HEAD);
                yourList->HEAD=tmpNode;
        }
               yourList->HEAD=NULL;
               yourList->TAIL=NULL;
               yourList->lineNum=0;
}

/* ****"p" - print**** */
void print(whole_list *fileBuffer,char *fileName){
        printf("%s\n",fileName);
        node *i;
        int count;
        count=0;

        for(i = fileBuffer->HEAD; i != NULL; i=i->next){
                if(count<9){
                        count++;
                        printf("0%i %s",count,i->content);
                }
                else{
                        count++;
                        printf("%i %s",count,i->content);
                }
        }
}

/* ****"a" - add function**** */
void add(whole_list *fileBuffer,int atLine,char *restOfString){

        node *i;
        int counter;
        counter = 1;

        /* Creating a new node for adding into our list */
        node *newNode;
        newNode = (node *)malloc(sizeof(node));
        strcpy(newNode->content,restOfString);
        newNode->next = NULL;

        if(atLine==0&&fileBuffer->HEAD==NULL&&fileBuffer->TAIL==NULL){
                newNode->next = fileBuffer->HEAD;
                fileBuffer->HEAD = newNode;
                fileBuffer->TAIL = newNode;
        }
        else if(atLine==0){
                newNode->next = fileBuffer->HEAD;
                fileBuffer->HEAD = newNode;
        }
        else{
                for(i=fileBuffer->HEAD;i !=NULL; i=i->next){
                        if(counter==atLine){
```

```
                                /* add */
                                newNode->next = i->next;
                                i->next = newNode;
                        }
                        counter++;
                }
                if(atLine==fileBuffer->lineNum){
                        /* update TAIL when adding to last node */
                        fileBuffer->TAIL=newNode;
                }
        }
        fileBuffer->lineNum=fileBuffer->lineNum+1;
}

/* "d" -- delete function */
/* d0 -- delete everything leaves 0 byte */
void delete(whole_list *fileBuffer, int atLine){

        node *i;
        node *prevNode;
        node *followNode;
        int counter;
        counter = 1;

        if(atLine==0){
                /* delete all */
                freeList(fileBuffer);
        }
        else{
                if(atLine==1){
                        /* delete the first line */
                        free(fileBuffer->HEAD);
                        fileBuffer->HEAD = fileBuffer->HEAD->next;
                }
                else if(atLine==fileBuffer->lineNum){
                        for(i=fileBuffer->HEAD; i!=NULL;i=i->next){
                                if(counter==fileBuffer->lineNum-1){
                                        free(i->next);
                                        i->next=NULL;
                                        fileBuffer->TAIL = i;
                                }
                                counter++;
                        }
                }
                else{
                        prevNode = fileBuffer->HEAD;
                        for(i=fileBuffer->HEAD->next; i!=NULL; i=i->next){
                                followNode = i->next;
                                if(counter==atLine-1){
                                        free(prevNode->next);
                                        prevNode->next = followNode;
                                }
                                counter++;
                                prevNode = i;
                        }
                }
                fileBuffer->lineNum--;
        } /* big else */
}

/* "r" -- replace function */
void replace(whole_list *fileBuffer,int atLine,char *restOfString){

        node *i;
        int counter;
        counter = 1;

        for(i=fileBuffer->HEAD;i!=NULL;i=i->next){
                if(counter==atLine){
                        strcpy(i->content,restOfString);
                }
```

```
                counter++;
        }
}

/* "s" -- save the modification */
void save(whole_list *fileBuffer, char *fileName){

        FILE * handler;
        handler = fopen(fileName,"w");
        if(handler==NULL){
                printf("Can't open file for writing\n");
                fclose(handler);
        }
        else{
                node *i;
                for(i=fileBuffer->HEAD;i!=NULL;i=i->next){
                        fprintf(handler,"%s",i->content);
                }
                fclose(handler);
        }
}

void blockInLast(whole_list *fileBuffer,whole_list *blockBuffer){
        /* method just for block insert at last line */
        if(fileBuffer->HEAD==NULL&&fileBuffer->TAIL==NULL){
                fileBuffer->HEAD = blockBuffer->HEAD;
                fileBuffer->TAIL = blockBuffer->TAIL;
        }
        else{
                fileBuffer->TAIL->next = blockBuffer->HEAD;
                fileBuffer->TAIL = blockBuffer->TAIL;
        }
}

/* "m" block insert*/
void blockInsert(whole_list *fileBuffer, whole_list *blockBuffer, int atLine){

        node *i;
        int counter;
        counter = 1;

        if(atLine==0){
                /* Insert at first */
                if(fileBuffer->HEAD!=NULL&&fileBuffer->TAIL!=NULL){
                        blockBuffer->TAIL->next = fileBuffer->HEAD;
                        fileBuffer->HEAD = blockBuffer->HEAD;
                }
                else{
                        blockInLast(fileBuffer,blockBuffer);
                }
        }
        else{
                for(i=fileBuffer->HEAD;i!=NULL;i=i->next){
                        if(counter==atLine){
                                blockBuffer->TAIL->next = i->next;
                                i->next = blockBuffer->HEAD;
                        }
                        counter++;
                }
        }
}

void popStr(whole_list *myList, char *tmp){
        node *i;
        int cntr;
        cntr=1;
        for(i=myList->HEAD;i!=NULL;i=i->next){
                if(cntr==myList->lineNum){
                        strcpy(tmp,i->content);
                }
                cntr++;
```

```
        }
        delete(myList,myList->lineNum);
}

/* "u" -- the undo function */
void undo(whole_list *fileBuffer, whole_list *cmdHistory, whole_list *histContent,
whole_list *backupBuffer){

        int tmpInt;
        int atLine;
        char cmd;

        char tmpCmd[MAX_CMDCHAR];
        char tmpStr[MAX_CHAR];

        node *j;
        node *h;
        int k;
    int cntr;
    cntr=1;

        popStr(cmdHistory,tmpCmd);
    cmd = tmpCmd[0];
    atLine = atoi(&tmpCmd[1]);

        switch(cmd){

                case 'a':       if(strlen(tmpCmd)==1){
                                                        delete(fileBuffer,fileBuffer-
>lineNum);
                                                }
                                                else{
                                                        delete(fileBuffer,atLine+1);
                                                }
                                                break;

                case 'r':               popStr(histContent,tmpStr);
                                                replace(fileBuffer,atLine,tmpStr);
                                                break;

                case 'd':       if(strcmp(tmpCmd,"d0\n")==0){
                                                /* undo DELETE ALL */
                                                /* POP the lineNumber */
                                        for(j=histContent->HEAD;j!=NULL;j=j->next){
                                                if(cntr==histContent->lineNum){
                                                        tmpInt = j->val;
                                                }
                                                cntr++;
                                        }
                                        delete(histContent,histContent->lineNum);
                                                                cntr = 1;

        for(k=1;k<=tmpInt;k++){
                                                                                /* POP
the backup content */
                                                for(h=backupBuffer->HEAD;h!=NULL;h=h->next){
                                                        if(cntr==backupBuffer->lineNum){
                                                                strcpy(tmpStr,h->content);
                                                        }
                                                        cntr++;
                                                }
                                                delete(backupBuffer,backupBuffer->lineNum);
                                                                                        cntr=1;

        add(fileBuffer,0,tmpStr);
                                                                        }
                                                                }
                                                                else{

        popStr(histContent,tmpStr);
```

```
                                                                        add(fileBuffer,atLine-
1,tmpStr);
                                                                }
                                                                break;

                case 'm':               /* POP content */
                                        for(j=histContent->HEAD;j!=NULL;j=j->next){
                                                if(cntr==histContent->lineNum){
                                                        tmpInt = j->val;
                                                }
                                                cntr++;
                                        }
                                        delete(histContent,histContent->lineNum);
                                        if(strcmp(tmpCmd,"m\n")==0){
                                                for(k=1;k<=tmpInt;k++){
                                                        delete(fileBuffer,fileBuffer->lineNum);
                                                }
                                        }
                                        else{
                                                for(k=1;k<=tmpInt;k++){
                                                        delete(fileBuffer,atLine+1);
                                                }
                                        }
                                        break;

                        default:                break;
        }
}

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

        /* ****variable declarations**** */
        int atLine;
        char fileName[MAX_FN_CHAR];
        char strTmp[MAX_CHAR];
        char stringBuffer[MAX_CHAR];
        char bufferCopy[MAX_CHAR];
        char *param;
        char *restOfString;
        char cmd;
        char blockString[MAX_CHAR];
        int blkLines;

        /* ****initialized fileBuffer/list instances**** */
        whole_list fileBuffer;
        fileBuffer.HEAD = NULL;
        fileBuffer.TAIL = NULL;
        fileBuffer.lineNum=0;

        whole_list blockBuffer;
        blockBuffer.HEAD = NULL;
        blockBuffer.TAIL = NULL;
        blockBuffer.lineNum=0;
        blkLines=0;

        /* ****undo lists**** */
        whole_list cmdHistory;
        cmdHistory.HEAD = NULL;
        cmdHistory.TAIL = NULL;
        cmdHistory.lineNum=0;

        whole_list histContent;
        histContent.HEAD=NULL;
        histContent.TAIL=NULL;
        histContent.lineNum=0;

        whole_list backupBuffer;
        backupBuffer.HEAD=NULL;
        backupBuffer.TAIL=NULL;
        backupBuffer.lineNum=0;
```

```
/* checking command line arguments */
if (argc == 3) {
        if(strcmp(argv[1],"-c")==0){
                FILE * checker;
                checker = fopen(argv[2],"r");
                if(checker==NULL){
                        fclose(checker);
                        strcpy(fileName, argv[2]);
                FILE * handler;
                handler = fopen(argv[2],"w");
        fclose(handler);
                }
                else{
                        fclose(checker);
                        printf("The specified file already exists\n");
                        exit (1);
                }
        }
        else{
                printf("Usage: ./simted3 [-c] foo.txt\n");
                exit (1);
        }
}
else if(argc == 2){
        /* simted3.c myFile.txt */
        /* open existing file to edit */
        strcpy(fileName,argv[1]);

        FILE * handler;
        handler = fopen(argv[1],"r");
        if((handler=fopen(argv[1],"r")) == NULL){
                fclose(handler);
                printf("The specified file does not exist\n");
                exit (1); /* exit with error */
        }
        else{
                while(fgets(strTmp,MAX_CHAR,handler)){
                        read(&fileBuffer,strTmp);
                        fileBuffer.lineNum=fileBuffer.lineNum+1;
                }
                fclose(handler);
        }
}
else{
        printf("Usage: ./simted3 [-c] foo.txt\n");
        exit (1);
}
while(1){
        printf(":");
        fgets(stringBuffer, MAX_CHAR, stdin);
        strcpy(bufferCopy,stringBuffer);

        /* ****string token***** */
        param = strtok(stringBuffer, " ");
        do{
                restOfString = strtok(NULL,"");

        }while(strtok(NULL," ") != NULL);
        atLine = atoi(&param[1]);
        cmd = param[0];

        switch(cmd){

                case 'a':       add(&cmdHistory,cmdHistory.lineNum,param);
                                                        if(strlen(param)==1){

add(&fileBuffer,fileBuffer.lineNum,restOfString);
                                                        }
                                                        else{
```

```
        add(&fileBuffer,atLine,restOfString);
                                                        }
                                                        break;

                        case 'd':       if(atLine==0){
                                                                /* push commands onto
stack */

        add(&cmdHistory,cmdHistory.lineNum,param);

                                                                /* saving the line
number for the original buffer */

        addInt(&histContent,histContent.lineNum,fileBuffer.lineNum);

                                                                /* saving fileBuffer
*/
                                                                node *i;

        for(i=fileBuffer.HEAD;i!=NULL;i=i->next){

        add(&backupBuffer,backupBuffer.lineNum,i->content);
                                                                }
                                                        }
                                                        else{

        add(&cmdHistory,cmdHistory.lineNum,param);
                                                                node *i;
                                                                int cntr;
                                                                cntr=1;

        for(i=fileBuffer.HEAD;i!=NULL;i=i->next){

        if(atLine==cntr){

        add(&histContent,histContent.lineNum,i->content);
                                                                        }
                                                                cntr++;
                                                                }
                                                        }
                                                        delete(&fileBuffer,atLine);
                                                        break;

                        case 'r':               add(&cmdHistory,cmdHistory.lineNum,param);
                                                        node *i;
                                                        int cntr;
                                                        cntr=1;

        for(i=fileBuffer.HEAD;i!=NULL;i=i->next){
                                                                if(atLine==cntr){

        add(&histContent,histContent.lineNum,i->content);
                                                                }
                                                                cntr++;
                                                        }

        replace(&fileBuffer,atLine,restOfString);
                                                        break;

                        case 'm':       add(&cmdHistory,cmdHistory.lineNum,param);
                                                        while(1){

        fgets(blockString,MAX_CHAR,stdin);


        if(strcmp(blockString,"\x0b\n")==0){
                                                                        break;
                                                                }
                                                                else{
```

```
        read(&blockBuffer,blockString);

        blockBuffer.lineNum++;

        fileBuffer.lineNum=fileBuffer.lineNum+1;
                                                                        }
                                                                }
                                                                /* Linking my block lists and
my orginal list */

        if(strcmp(bufferCopy,"m\n")==0){
                                                                        /* insert at last line
*/

        blockInLast(&fileBuffer,&blockBuffer);
                                                                }
                                                                else{

        blockInsert(&fileBuffer,&blockBuffer,atLine);
                                                                }

        addInt(&histContent,histContent.lineNum,blockBuffer.lineNum);
                                                                blockBuffer.HEAD = NULL;
                                                                blockBuffer.TAIL = NULL;
                                                                blockBuffer.lineNum=0;
                                                                break;

                        case 'p':       if(strcmp(bufferCopy,"p\n")==0){

        print(&fileBuffer,fileName);
                                                                }
                                                                break;

                        case 's':       if(strcmp(bufferCopy,"s\n")==0){

        save(&fileBuffer,fileName);
                                                                        freeList(&cmdHistory);
                                                                       freeList(&histContent);

        freeList(&backupBuffer);
                                                                       freeList(&blockBuffer);
                                                                }
                                                                break;

                        case 'q':       if(strcmp(bufferCopy,"q\n")==0){
                                                                        /* ****save and quit
program**** */

        save(&fileBuffer,fileName);
                                                                        exit (0);
                                                                }
                                                                else
if(strcmp(bufferCopy,"qns\n")==0){
                                                                        /* ****quit with out
save**** */
                                                                        exit (0);
                                                                }
                                                                else {
                                                                        break;
                                                                }

                        case 'u':       if(cmdHistory.lineNum==0){
                                                                        printf("Cannot
Undo\n");
                                                                }
                                                                else{

        undo(&fileBuffer,&cmdHistory,&histContent,&backupBuffer);
                                                                }
                                                                break;
```

```
                        default:                break;
                } /* ****closing of switch*/
        }
}
```