```c
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <string.h>

//https://stackoverflow.com/questions/37245312/single-producer-and-multiple-consumers


char exitChar[4] = "quit";
char buf[200];
int done;

//Consumer data struct
struct consumer_state {
        int value_available;
        char* buf;
        pthread_mutex_t mutex;
        pthread_cond_t cond;
} cons_state;

//Printer data struct
struct printer_state {
        int printing;
        char* buf;
        int printing_thread_id;
        pthread_mutex_t mutex;
        pthread_cond_t cond;
        pthread_cond_t main_cond;
} print_state;


//Printer thread
void *printerThread(void *args){

        //While !quit
        while(!done){

                //printf("Locking mutex @ printer...\n\n");
                pthread_mutex_lock(&print_state.mutex);
                //printf("Mutex locked @ printer...\n\n");

                //While theres no input to print available, wait
                if(!cons_state.value_available)
                        pthread_cond_wait(&print_state.cond,&print_state.mutex);

                if(!done){
                        printf("\n%i: %s", print_state.printing_thread_id,print_state.buf);
                        //Reset variables after printing
                        print_state.printing = 0;
                        cons_state.value_available = 0;
                        print_state.printing_thread_id = 9;
                        //Signal mainline printing is finished
                        pthread_cond_signal(&print_state.main_cond);
                }
                //Unlock mutex
                pthread_mutex_unlock(&print_state.mutex);
        }
        free(args);
        pthread_exit(0);
}

//Consumer thread
void *consumerThread(void *args){

        int threadId = *((int *) args);
        //While !quit
        while(!done){
                pthread_mutex_lock(&cons_state.mutex);

                //While theres no input available, wait
                if(!cons_state.value_available)
                        pthread_cond_wait(&cons_state.cond,&cons_state.mutex);

                //When condition signal recieved & text != quit, signal printer to print
                if(!done){
                        print_state.printing_thread_id = threadId;
                        print_state.buf = cons_state.buf;
                        pthread_cond_signal(&print_state.cond);
                }
                pthread_mutex_unlock(&cons_state.mutex);
        }
        free(args);
        pthread_exit(0);
}

int main( ) {

        int done = 0;
        pthread_t consumer_threads[3];
        pthread_t printer_thread;
        int t, returnCode, carryOn, i;

        //Initialize consumer struct variables
        cons_state.value_available = 0;
        cons_state.mutex = (pthread_mutex_t)PTHREAD_MUTEX_INITIALIZER;
        cons_state.cond = (pthread_cond_t)PTHREAD_COND_INITIALIZER;

        //Initialize printer struct variables
        print_state.printing = 0;
        print_state.printing_thread_id = 9;
        print_state.mutex = (pthread_mutex_t)PTHREAD_MUTEX_INITIALIZER;
        print_state.cond = (pthread_cond_t)PTHREAD_COND_INITIALIZER;
        print_state.main_cond = (pthread_cond_t)PTHREAD_COND_INITIALIZER;

        //Create consumer threads
        for(t=1;t<=3;t++){

                //printf("Creating consumer thread number %i\n\n",t);
                int *arg = malloc(sizeof(*arg));
                *arg = t;
                //Create integration thread and pass arguments for current interval
                returnCode = pthread_create(&consumer_threads[t],NULL,consumerThread,arg);
                if (returnCode) {
                        printf("ERROR return code from pthread_create() at consumerThread: %d\n",returnCode);
                        exit(-1);
                }
        }


        //Create printer thread
        //printf("Creating printer thread...\n\n");
        returnCode = pthread_create(&printer_thread,NULL,printerThread,NULL);
        if (returnCode) {
                printf("ERROR return code from pthread_create() at consumerThread: %d\n",returnCode);
                exit(-1);
        }

        carryOn = 1;
    printf("Enter 'quit' to exit program...\n\n");
    printf("Enter a string: ");

        while(!done) {

                //Get string from user
                fgets(buf, 200, stdin);
                //printf( "\nYou entered: %s", buf);

                /* remove newline, if present */
                i = strlen(buf)-1;
                if( buf[ i ] == '\n')
                        buf[i] = '\0';

                //If input = quit
        if(strcmp(buf, exitChar) == 0){
                        carryOn = 0;
                        done = 1;
                        //This allows threads to exit
                        pthread_cond_broadcast(&cons_state.cond);
                        pthread_cond_signal(&print_state.cond);
                        printf("Goodbye!\n");
                }

                if(!done) {
                        //Tell any waiting consumer to consume
                        pthread_mutex_lock(&cons_state.mutex);
                        cons_state.buf = strdup(buf);
                        cons_state.value_available = 1;
                        print_state.printing = 1;
                        pthread_cond_signal(&cons_state.cond);

                        //While printing wait
                        if(print_state.printing)
                                pthread_cond_wait(&print_state.main_cond,&cons_state.mutex);

                        //Get next string
                        printf("\n\nEnter a string: ");
                        pthread_mutex_unlock(&cons_state.mutex);
                }
        }


        for(t=1;t<=3; t++)
        pthread_join(consumer_threads[t], NULL);

    pthread_join(printer_thread,NULL);
    printf("Successfully exited all threads!\n");
}
```