```
1  /**
2   * @file hwk.c
3   * @author Deepika Rajarajan,Fnu Deepak Ramadass,Pooja BurlyPrakash
4   * @brief This file contains all the functions of the program
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <stdint.h>
10 #include <assert.h>
11 #include <string.h>
12 #include <ctype.h>
13 #include "hwk.h"
14 #include "common.h"
15 #define SIZE 1000 ///< the max length of line
16 #define LOG(cmd) fprintf(log_file,"%s ERROR INVALID INPUT\n",strtok(cmd,"\r\n")) //
   /< Logs invalid commands
17 #define LOG_FUNDS(cmd) fprintf(log_file,"%s INSUFFICIENT FUNDS\n",strtok(cmd,"\r\n"
   )) ///< logs insufficient funds
18
19 int64_t FEE = 0; ///< store the FEE
20 struct STOCKS *database = NULL;
21 int64_t *user_data;
22 //char received_cmd[10];
23 char str[] = "\0"; ///< to find end of arrays;
24 char line_store[SIZE] = {}; ///< temp store of line for operation
25 FILE *log_file; ///< points to log file
26 char line[SIZE]; ///< stores the line
27 char log_line[SIZE]= {}; ///< store line for logging
28 char *ip; ///< loopback ip
29 char *port; ///< 10689 port used
30 char output1[SIZE];///< to store output from number to roman
31 char nameinp[4]; ///< to store stock name
32 /**
33  * Opens the file
34  */
35 void open_file(){
36         log_file = fopen("log.txt","w");
37         if(log_file == NULL){
38                 printf("Unable to open log file");
39                 exit(1);
40         }
41 }
42
43 /**
44  * Closes the the file
45  */
46 void close_file(){
47         fclose(log_file);
48 }
49
50 /**
51  * Error function
52  * @return 1
53  */
54 int error(){
55         printf("ERROR INVALID INPUT\n");
56         return 1;
57 }
58
59 /**
60  * Function that returns number of characters in the roman numeral
61  * @param dst input data
```

```
 * @return count return the number of characters
 */
uint8_t num_units_quanlen(char *dst)
{

        int i;
        int count = 0;
        for (i=0;dst[i]!='\0';i++)
        {
        count++;
        }
return count;
}
/**
 * Function that prints No shares available for purchase
 * @return 1
 */
int no_shares()
{
        printf("NO SHARES AVAILABLE FOR PURCHASE\n");
        return 1;
}
/**
 * Function that prints Communication error
 * @return 1
 */

int communication_error()
{
        printf("COMMUNICATION ERROR\n");
        return 1;
}
 /**
 * Register handler for client
 */

void client_register_handler()
{
        struct sigaction actinfo;
        actinfo.sa_handler = handler;
        sigfillset(&actinfo.sa_mask);
        actinfo.sa_flags = 0;
        sigaction(SIGALRM, &actinfo, 0);
}

/**
 * Throws Insufficient funds at console
 * @return 1
 */
int insuff_funds(){
        printf("INSUFFICIENT FUNDS\n");
        return 1;
}

/**
 * initializes my array of numbers
 * @param num the number of numbers
 * @return 1 if failure, 0 if success
 */
int initialize(size_t num)
{
        database = malloc(sizeof(struct STOCKS));
        if (database == NULL)
```

```
125         {
126                 printf("uh oh\n");
127                 return 1;
128         }
129 
130         database->num_values = num;
131         database->values = malloc(sizeof(int64_t) *(num));
132         database->value_names = malloc(sizeof(char *) * (num));
133         user_data = calloc(num+1,sizeof(int64_t));
134         user_data[0] = 0;
135         if ( database->values == NULL && database->value_names == NULL && user_d
    ata == NULL)
136         {
137                 printf("uh oh\n");
138                 return 1;
139         }
140         return 0;
141 }
142 
143 /**
144  * cleans up my array of numbers
145  */
146 void database_cleanup()
147 {
148         if (database == NULL || database->values == NULL || database->value_name
    s == NULL || user_data == NULL)
149         {
150                 printf("uh oh\n");
151         }
152         free(database->values);
153         database->values = NULL;
154         int count = (database->num_values) - 1;
155         while(count >= 0){
156                 free(database->value_names[count]);
157                 count--;
158         }
159         free(database->value_names);
160         database->value_names = NULL;
161         database->num_values = 0;
162         free(user_data);
163         user_data = NULL;
164         free(database);
165         database = NULL;
166 }
167 
168 /**
169   * Function used to store stock names
170   * @param token holds the stock name
171   * @param lines holds the position of the stock in the array
172 */
173 void store_name(char* token, int lines){
174         database->value_names[lines] = strdup(token);
175 }
176 
177 /**
178   * Function used to store stock value
179   * @param value holds the stock value
180   * @param lines holds the position of the stock in the array
181 */
182 void store_value(int64_t value, int lines){
183         (*database).values[lines] = value;
184 }
185 
```

```
186 /**
187  * Function to buy stocks from the loaded stock database
188  * @param stockname[] name of the stock
189  * @param num_units is the number of stocks to be bought
190  * @return 1 if failure, 0 if success
191  */
192 int buy(char stockname[],int num_units)
193 {
194         int64_t total;
195         int64_t stockprice;
196         int i=0;
197         int priceval = 0;
198         int64_t old_price;
199         int64_t new_price;
200 //Check if Num of stocks is 0 and return Error
201         if(database->num_values==0)
202         {
203         error();
204         LOG(log_line);
205         fflush(log_file);
206         return 1;
207         }
208 //Check if the received stockname is matching with the stocknames in user databa
    se
209 //Storing number of units to be bought in user database
210         for(i=0;i<database->num_values;i++)
211         {
212                 if(strcmp(stockname,database->value_names[i])==0)
213                 {
214                 stockprice = database->values[i];
215                 user_data[i+1]+=num_units;
216                 priceval = i;
217                 goto label4;
218                 }
219         }
220         error();
221         LOG(log_line);
222         fflush(log_file);
223         return 1;
224 label4:     total = num_units*stockprice;
225 //Check if user balance is not equal to 0 else print insufficient funds
226         if(user_data[0]!=0)
227         {
228 //Check if the shares to be bought is less than balance+fee
229                 if((total)<=user_data[0]-8)
230                 {
231 //Check if user balance is >= 10000 for no fee charge and reduce price of stock
    to $1
232                         if(user_data[0]>=10000)
233                         {
234
235                         user_data[0] = user_data[0]-(total);
236                         }
237 //User balance less than 10000 fee charge of $8 and reduce the price od stock to
     $1
238                         else
239                         {
240                         FEE = 8;
241                         user_data[0] = user_data[0]-(total)-FEE;
242                         }
243                 }
244 //Check insufficient funds
245                 else
```

```
246                 {
247                 insuff_funds();
248                 LOG_FUNDS(log_line);
249                 fflush(log_file);
250                 return 1;
251                 }
252         }
253         else
254         {
255         insuff_funds();
256         LOG_FUNDS(log_line);
257         fflush(log_file);
258         return 1;
259         }
260
261
262         old_price = database->values[priceval];
263         database->values[priceval] = database->values[priceval]-1;
264 //Check if stock price hits 0 and print error
265         if(database->values[priceval]<0)
266         {
267         database->values[priceval]=0;
268         error();
269         LOG(log_line);
270         fflush(log_file);
271         return 1;
272         }
273         new_price = database->values[priceval];
274         printf("%d SHARES OF %s BOUGHT FOR $%lld TOTAL ~~~ BALANCE NOW $%lld\n",num_u
    nits,stockname,total,user_data[0]);
275         fprintf(log_file,"BUY %s %d BALANCE $%lld FEE $%lld %s %lld $%lld $%lld\n",stockn
    ame,num_units,user_data[0],FEE,stockname,user_data[priceval+1],old_price,new_pri
    ce);
276         fflush(log_file);
277 return 0;
278 }
279
280 /**
281  * Deposits the amount specified by the user and adds to the user balance
282  *@param dollars amount of money to be deposited
283  *@return Function returns 0 if success
284  */
285 int deposit (int64_t dollars)
286 {
287 //add amount specified by the user to the user balance
288 //and display the current user balance
289         user_data[0] = user_data[0] + dollars;
290         printf("$%lld DEPOSITED ~~~ BALANCE NOW $%lld\n",dollars,user_data[0]);
291         fprintf(log_file,"DEPOSIT $%lld BALANCE $%lld FEE $%lld ----\n",dollars,user_
    data[0],FEE);
292         fflush(log_file);
293         return 0;
294
295 }
296
297 /**
298  * Withdraws the amount specified by the user from the user balance database
299  * @param dollars Amount of dollars to be withdrawn
300  * @return 1 if failure, 0 if success
301  */
302
303 int withdraw (int64_t dollars)
304 {
```

```
305 //Check if user balance is not 0
306         if(user_data[0]!=0)
307         {
308 //Check if withdraw amount is less than or equal to balance&fee
309                 if(dollars<=user_data[0]-10)
310                 {
311 //Check if user balance is > $10000 for no fee charge
312                         if(user_data[0]>=10000)
313                         {
314                         user_data[0] = user_data[0]-dollars;
315                         printf("HERE IS YOUR $%lld ~~~ BALANCE NOW $%lld\n",dollars,
    user_data[0]);
316                         fprintf(log_file,"WITHDRAW BALANCE $%lld FEE $%lld ----\n",
    user_data[0],FEE);
317                         fflush(log_file);
318                         }
319 //Check if user balance is < $10000 for fee charge of $10
320                         else if(user_data[0]<10000)
321                         {
322                         FEE = 10;
323                         user_data[0] = (user_data[0]-(dollars)-FEE);
324                         printf("HERE IS YOUR $%lld ~~~ BALANCE NOW $%lld\n",dollars,
    user_data[0]);
325                         fprintf(log_file,"WITHDRAW BALANCE $%lld FEE $%lld ----\n",
    user_data[0],FEE);
326                         fflush(log_file);
327                         }
328                 }
329                 else
330                 {
331                 insuff_funds();
332                 LOG_FUNDS(log_line);
333                 fflush(log_file);
334                 return 1;
335                 }
336         }
337         else
338         {
339         insuff_funds();
340         LOG_FUNDS(log_line);
341         fflush(log_file);
342         return 1;
343         }
344 return 0;
345 }
346
347 /** Quotes the current stock price of a stock specified by the user
348  * @param stock_name[] User specified stock name
349  * @return returns 1 in case of any error and 0 if success
350  */
351 int quote (char stock_name[])
352 {
353         int64_t i = 0;
354         int64_t old_price = 0;
355         int64_t fee_amt = 0;
356         int pos = 0;
357 //Check if number of stocks is 0
358         if(database->num_values==0)
359         {
360         LOG(log_line);
361         error();
362         fflush(log_file);
363         return 1;
```

```
364         }
365 //Check if user balance is greater than 1 else print insufficient funds
366         if (user_data[0] > 1)
367         {
368
369                 for (i = 0; i<database->num_values; i++)
370                 {
371 //checks if the user has entered a valid stock name
372                         if (strcmp(stock_name,database->value_names[i]) == 0)
373
374                         {
375                                 pos = i;
376                                 old_price = database->values[i];
377 //checks if the user owns any stocks for the entered stock name
378                                 if (user_data[i+1] == 0)
379                                 {
380                                         database->values[i] = database->values[i
    ] + 1;
381                                 }
382                                 else if (user_data[i+1] >= 1)
383                                 {
384                                         database->values[i] = database->values[i
    ] - 1;
385                                         if (database->values[i] < 0)
386                                                 database->values[i] = 0;
387                                 }
388                                 goto label5;
389                         }
390                 }
391                 LOG(log_line);
392                 fflush(log_file);
393                 error();
394                 return 1;
395 //checks if the user has $10000 or more in their account to determine the fees
396                label5:  if ((user_data[0] + database->values[pos]) >= 10000)
397                 {
398                         fee_amt = 0;
399                 }
400                 else
401                 {
402                         fee_amt = 1;
403                 }
404                 user_data[0] = user_data[0] - fee_amt;
405                 printf("%s IS $%lld\n",stock_name,database->values[pos]);
406                 if (old_price == 0)
407                 {
408                         fprintf(log_file,"QUOTE %s BALANCE $%lld FEE $%lld %s - $%lld
    -\n",stock_name,user_data[0],fee_amt,stock_name,old_price);
409                 }
410                 else
411                 {
412                         fprintf(log_file,"QUOTE %s BALANCE $%lld FEE $%lld %s - $%lld
    $%lld\n",stock_name,user_data[0],fee_amt,stock_name,old_price,database->values[po
    s]);
413                 }
414                 fflush(log_file);
415         }
416         else
417         {
418                 LOG_FUNDS(log_line);
419                 fflush(log_file);
420                 insuff_funds();
421                 return 1;
```

```
422         }
423         return 0;
424 }
425
426 /**
427   * The sell fuctions checks if the user balance is above 10000
428   * and charges fee accodingly, it also checks if the stock is bankrupy
429   * and prints an error is sell is used on a bankrupt stock
430   * It sell stocks, add the sale to balance, increases stock price by 1 and redu
    ces no.of stocks owned
431   * @param name is the stock name
432   * @param units is number of stocks to be sold
433   * @return 0/1 - success/error
434 */
435 int sell(char *name, int units){
436         if(user_data[0] >= 10000)
437                 FEE = 0;
438         else
439                 FEE = 8;
440         int count = 0;
441         while( count < database->num_values){
442                 if(strcmp(name,(database->value_names[count])) == 0)
443                         break;
444                 count++;
445         }
446         if(count >= database->num_values){
447                 LOG(log_line);
448                 error();
449                 fflush(log_file);
450                 return 1;
451         }
452         int64_t price = database->values[count];
453         if(price <= 0){
454                 error();
455                 LOG(log_line);
456                 fflush(log_file);
457                 return 1;
458         }
459         if(units > user_data[count+1]){
460                 LOG(log_line);
461                 error();
462                 fflush(log_file);
463                 return 1;
464         }
465         user_data[0]+= (price * units);
466         if(user_data[0] < FEE){
467                 LOG(log_line);
468                 error();
469                 fflush(log_file);
470                 return 1;
471         }
472         (database->values[count])++;
473         user_data[0]-= FEE;
474         user_data[count+1]-= units;
475         printf("%d SHARES OF %s SOLD FOR $%lld TOTAL ~~~ BALANCE NOW $%lld\n",units,n
    ame,(price * units),user_data[0]);
476         fprintf(log_file,"SELL %s %d BALANCE $%lld FEE $%lld %s %lld $%lld $%lld\n",name,u
    nits,user_data[0],FEE,name,user_data[count+1],price,price+1);
477         fflush(log_file);
478         return 0;
479 }
480
481 /**
```

```
482   * The statement function prints the total balance,individual stocks and prices
483   * reduces the fee accordingly
484   * @return 0/1 - success/error
485 */
486 int statement(){
487         if(user_data[0] >= 10000)
488                 FEE = 0;
489         else
490                 FEE = 2;
491         int64_t funds_check = user_data[0] - FEE;
492         if(funds_check < 0){
493                 LOG_FUNDS(log_line);
494                 fflush(log_file);
495                 insuff_funds();
496                 return 1;
497         }
498         printf("BALANCE NOW $%lld\n",(user_data[0] - FEE));
499         printf("STOCK\tSHARES\tPRICE\tTOTAL\n");
500         int num = (database->num_values)-1;
501         int64_t TOTAL = 0;
502         int count = 0;
503         while(count <= num){
504                 int64_t units = user_data[count+1];
505                 if(units == 0){
506                         count++;
507                         continue;
508                 }
509                 int64_t price = database->values[count];
510                 if(price <= 0){
511                         count++;
512                         continue;
513                 }
514                 int64_t total = units * price;
515                 TOTAL+= total;
516                 printf("%s\t%lld\t$%lld\t$%lld\n",(database->value_names[count]),units
    ,price,total);
517                 count++;
518         }
519         TOTAL = TOTAL + user_data[0] - FEE;
520         if(count > num)
521                 printf("ACCOUNT VALUE $%lld\n",TOTAL);
522         user_data[0]-= FEE;
523         fprintf(log_file,"STATEMENT BALANCE $%lld FEE $%lld - - - -\n",user_data[0],F
    EE);
524         fflush(log_file);
525         return 0;
526 }
527
528 /**
529   * Funtions finds the number of strings in a line
530   * @param line points to the line
531   * @return tokens - the number of strings
532 */
533 int check_arguments(char* line){
534         char *token = strtok(line," ");
535         int tokens = 0;
536         while(token != NULL){
537                 token = strtok(NULL, " ");
538                 tokens++;
539         }
540         return tokens;
541 }
542
```

```
543 /**
544  * Checks the capitalization of the input string given
545  * @param stock[] stockname
546  * @return 1 if failure, 0 if success
547  */
548 int cmdcaps(char stock[])
549 {
550         int i;
551         for(i=0;stock[i]!='\0';i++)
552         {
553         if(stock[i]>='A' && stock[i]<='Z')
554         {
555         }
556         else
557         {
558         return 1;
559         }
560         }
561 return 0;
562 }
563
564 /**
565  * Validation for string of numbers
566  * @param numinput character array of numbers
567  * @return 1 if failure, 0 if success
568  */
569
570 int numcheck(char numinput[])
571 {
572 int i;
573         for(i=0;numinput[i]!='\0';i++)
574         {
575                 if((numinput[i]>='0') && (numinput[i]<='9'))
576                 {
577                 }
578                 else
579                 {
580                 return 1;
581                 }
582         }
583 return 0;
584 }
585
586 /**
587  * Coverts to uppercase
588  * @param data character array
589  * @return 0 if success
590  */
591
592 int uppercase(char data[])
593 {
594         int i =0;
595         for(i=0;data[i]!='\0';i++)
596         {
597         data[i] = toupper(data[i]);
598         }
599         return 0;
600 }
601 /**
602  * Enters into transaction mode
603  * Validates the input arguments and checks the validation of each argument
604  * Calls the transaction functions BUY,SELL,DEPOSIT,WITHDRAW,STATEMENT,QUOTE
605  * @param ip ip for transfer
```

```
606  * @param port port for transfer
607  * @return 1 if failure, 0 if success
608  */
609 
610 int transaction_mode(char* ip,char* port)
611 {
612         char line[SIZE]={};
613         int num_units=0;
614         int64_t dollars=0;
615         int arg_check=0;
616         printf("TRANSACTION MODE READY\n");
617         label3: if(fgets(line,SIZE,stdin)== NULL)
618         {
619                 return 1;
620         }
621         strcpy(log_line,line);
622         strcpy(line_store,line);
623         arg_check = check_arguments(line_store);
624         char input1[SIZE]={0};
625         char input2[SIZE] ={0};
626         char input3[SIZE] = {0};
627         uint8_t quanlen;
628         sscanf(line_store,"%s",input1);
629 //Checks if transaction command is in caps
630                 if(cmdcaps(input1)!=0)
631                 {
632                 LOG(log_line);
633                 error();
634                 fflush(log_file);
635                 goto label3;
636                 }
637 //Checks if buy or sell transaction
638                 if(strcmp(input1,"BUY")==0 || strcmp(input1,"SELL")==0)
639                 {
640                         if(arg_check != 3)
641                         {
642                         LOG(log_line);
643                         error();
644                         fflush(log_file);
645                         goto label3;
646                         }
647                         sscanf(line,"%s %s %s",input1,input2,input3);
648                          if(cmdcaps(input2)!=0 || input2[4]!=0 || numcheck(input
    3)!=0)
649                         {
650                                 LOG(log_line);
651                                 error();
652                                 fflush(log_file);
653                                 goto label3;
654                         }
655                         num_units = atoi(input3);
656                         if(num_units>100)
657                         {
658                         printf("Only 100 shares may be bought or sold\n");
659                         goto label3;
660                         }
661 //Buy transaction
662                         if(strncmp(input1,"BUY",3)==0)
663                         {
664 //Check if the received stockname is matching with the stocknames in user databa
    se
665                         int k=0;
666                         int count = 0;
```

```
667                             int64_t total1 = 0;
668                             int64_t stockprice1 = 0;
669                             for(k=0;k<database->num_values;k++)
670                             {
671                             if(strncmp(input2,database->value_names[k],4)==0)
672                             {
673                             count=count+1;
674                             stockprice1= database->values[k];
675                             goto label6;
676                             }
677                             }
678                     label6: if(count!=1)
679                             {
680                             LOG(log_line);
681                             error();
682                             fflush(log_file);
683                             goto label3;
684                             }
685     //Check insufficient funds
686                             total1 = stockprice1*num_units;
687                             if((user_data[0]==0) || (total1>user_data[0]-8))
688                             {
689                             insuff_funds();
690                             LOG_FUNDS(log_line);
691                             fflush(log_file);
692                             goto label3;
693                             }
694                             if(stockprice1 == 0)
695                             {
696                             LOG(log_line);
697                             error();
698                             fflush(log_file);
699                             goto label3;
700                             }
701
702     //Convert the units got from the user to roman numeral
703                             number_to_roman(output1,num_units);
704     //Convert the roman numeral to uppercase
705                             uppercase(output1);
706     //Call function num_units_quanlen to find out the number of characters in roman
        numeral
707                             quanlen = num_units_quanlen(output1);
708                             int i;
709                             char nameinp[4];
710                             char input2copy[SIZE]={};
711                             char *nameinp2;
712                             strcpy(input2copy,input2);
713                             nameinp2 = strtok(input2copy," ");
714     //Stock name when a character is Null replaces with whitespace character
715                             for(i=0;i<4;i++)
716                             {
717                             if(nameinp2[i]=='\0')
718                             {
719                             nameinp2[i] =' ';
720                             }
721                             else
722                             {
723                             nameinp2[i]=nameinp2[i];
724                             }
725                             }
726                             strncpy(nameinp,nameinp2,4);
727                             int clientresult=0;
728     //Check result from client
```

```
729                         clientresult = client(nameinp,output1,quanlen,ip,port);
730                         if (clientresult == 0)
731                         {
732                         buy(input2,num_units);
733                         goto label3;
734                         }
735                         else
736                         {
737                         goto label3;
738                         }
739                         }
740 //Sell transaction
741                         if(strcmp(input1,"SELL")==0)
742                         {
743                         sell(input2,num_units);
744                         goto label3;
745                         }
746
747                 }
748 //Checks deposit and withdraw transaction
749                 else if(strcmp(input1,"DEPOSIT")==0 || strcmp(input1,"WITHDRAW
    ")==0)
750                 {
751
752                         if(arg_check!=2)
753                         {
754                         LOG(log_line);
755                         error();
756                         fflush(log_file);
757                         goto label3;
758                         }
759                         sscanf(line,"%s %s",input1,input2);
760                         char *stockinp;
761                         stockinp = input2+1;
762                         if(input2[0]!='$'||numcheck(stockinp)!=0)
763                         {
764                         LOG(log_line);
765                         error();
766                         fflush(log_file);
767                         goto label3;
768                         }
769                         dollars = atoi(stockinp);
770 //Deposit transaction
771                         if(strcmp(input1,"DEPOSIT")==0)
772                         {
773                         deposit(dollars);
774                         goto label3;
775                         }
776 //Withdraw transaction
777                         if(strcmp(input1,"WITHDRAW")==0)
778                         {
779                         withdraw(dollars);
780                         goto label3;
781                         }
782
783                 }
784 //Checks quote transaction
785                 else if(strcmp(input1,"QUOTE")==0)
786                 {
787                         if(arg_check!=2)
788                         {
789                         LOG(log_line);
790                         error();
```

```
                        fflush(log_file);
                        goto label3;
                        }
                        sscanf(line,"%s %s",input1,input2);
                                if(cmdcaps(input2)!=0)
                                {
                                LOG(log_line);
                                error();
                                fflush(log_file);
                                goto label3;
                                }
                                else
                                {
                                quote(input2);
                                goto label3;
                                }
                }
//Checks statement transaction
                else if(strcmp(input1,"STATEMENT")==0)
                {
                        if(arg_check!=1)
                        {
                        LOG(log_line);
                        error();
                        fflush(log_file);
                        goto label3;
                        }
                        sscanf(line,"%s",input1);
                        statement();
                        goto label3;
                }
                else
                {
                LOG(log_line);
                error();
                fflush(log_file);
                goto label3;
                }
return 0;
}

/**
  * This function checks the initial STOCK commands and
  * intializes the database and store the stock names and values
  * it checks if the STOCKS/STOCKS command is present first and
  * then the approprite pattern STOCKS Num/ STOCK xyx value
  * It throws an error is Num/ value is <= 0 and expects xyx to be uppercase.
  * @param data gets ip and port
  * @return 0/1 if successful/failure
*/
int startup_mode(char* data[]){
        printf("STARTUP MODE READY\n");
        char line[SIZE];
        char cmd[SIZE];
        size_t num;
        int length = 0;
        int64_t value = 0;
        int result = 0;
        open_file();
label1: if(fgets(line,SIZE,stdin) == NULL){
                return 1;
        }
        strcpy(log_line,line);
```

```
854             strcpy(line_store,line);
855             if(check_arguments(line_store) != 2){
856                     LOG(log_line);
857                     error();
858                     fflush(log_file);
859                     goto label1;
860             }
861             if(sscanf(line,"%6s",cmd) == EOF){
862                     LOG(log_line);
863                     error();
864                     fflush(log_file);
865                     goto label1;
866             }
867             if(strcmp(cmd,"STOCKS") != 0){
868                     LOG(log_line);
869                     error();
870                     fflush(log_file);
871                     goto label1;
872             }
873             result = sscanf(line,"STOCKS %d",&length);
874             if(length <= 0){
875                     LOG(log_line);
876                     error();
877                     fflush(log_file);
878                     goto label1;
879             }
880             num = length;
881             if(result != 1){
882                     LOG(log_line);
883                     error();
884                     fflush(log_file);
885                     goto label1;
886             }
887             if(initialize(num) !=0){
888                     LOG(log_line);
889                     error();
890                     fflush(log_file);
891                     goto label1;
892             }
893             fprintf(log_file,"STOCKS %d BALANCE $0 FEE $0 - - - -\n",num);
894             fflush(log_file);
895
896             int lines = 0;
897             char tokens[4] = {};
898
899     label2: while(lines < num && fgets(line,SIZE,stdin) != NULL){
900                     strcpy(log_line,line);
901                     strcpy(line_store,line);
902                     if(sscanf(line,"%5s",cmd) == EOF){
903                             LOG(log_line);
904                             error();
905                             fflush(log_file);
906                             goto label2;
907                     }
908                     if(check_arguments(line_store) != 3){
909                             LOG(log_line);
910                             error();
911                             fflush(log_file);
912                             goto label2;
913                     }
914                     if(strcmp(cmd,"STOCK") != 0){
915                             LOG(log_line);
916                             error();
```

```
917                             fflush(log_file);
918                             goto label2;
919                     }
920                     //strcpy(line_store,line);
921                     if((result = sscanf(line,"STOCK %4[A-Z] $%lld",tokens,&value) != 2
    )){
922                             LOG(log_line);
923                             error();
924                             fflush(log_file);
925                             goto label2;
926                     }
927                     if(value <= 0){
928                             LOG(log_line);
929                             error();
930                             fflush(log_file);
931                             goto label2;
932                     }
933                     int count = 0;
934                     while(count < lines){
935                                     if(strncmp(tokens,(database->value_names[count])
    ,4) == 0){
936
937                                             LOG(log_line);
938                                             error();
939                                             fflush(log_file);
940                                             goto label2;
941                                     }
942                             count ++;
943                     }
944                     store_value(value,lines);
945                     store_name(tokens,lines);
946                     fprintf(log_file,"STOCK %s $%lld BALANCE $0 FEE $0 - - - -\n",tokens,va
    lue);
947                     fflush(log_file);
948                     lines++;
949             }
950     //      printf("data[1] = %s\n",data[1]);
951     //      printf("data[2] = %s\n",data[2]);
952             ip = data[1];
953             port = data[2];
954     //      printf("ip = %s\n",ip);
955     //      printf("port = %s\n",port);
956             transaction_mode(ip,port);
957             database_cleanup();
958             fclose(log_file);
959             return 0;
960     }
```

```
/**
 * @file main.c
 * @author Deepak Ramadass,Deepika Rajarajan
 * @brief This is the main file that starts the program.
 */
#include "hwk.h"
#include "common.h"
/**
 * This is the main function that starts the program.
 * @param argc the number of arguments
 * @param argv the command line arguments
 *   (the 0th argument is the program name)
 * @return  0 the exit code of the program
 */
int main(int argc, char *argv[])
{
        srandom(time(NULL));
        //Check if arguments are given else error is thrown into console
        if (argc >3 || argc <3)
        {
                fprintf(stderr,"Error: Please enter valid number of input arguments\n");
                exit(1);
        }
        else
        {
                if(argc==3)
                {
                struct addrinfo lookup_addr;
                memset(&lookup_addr, 0, sizeof(struct addrinfo));
                lookup_addr.ai_family = AF_INET6;
                lookup_addr.ai_socktype = SOCK_DGRAM;
                lookup_addr.ai_protocol = IPPROTO_UDP;
                struct addrinfo *send_addr;
                        if (getaddrinfo(argv[1],argv[2], &lookup_addr, &send_add
r)!=0)
                        {

                                perror("getaddrinfo failed");
                                printf("Invalid Ip or port\n");
                                exit(1);
                        }
                freeaddrinfo(send_addr);
                        /*if(lookup_addr.ai_family == AF_INET6)
                        {
                        printf("%s is an ipv6 address\n",argv[1]);
                        }
                        else
                        {
                        printf("%s is an is unknown address format\n",argv[1]);
                        exit(1);
                        }


                int i;
                printf("%s\n",argv[2]);
                for(i=0;strlen(argv[2]);i++)
                {
                        if((argv[2][i]>='0') && (argv[2][i] <='9'))
                                continue;
                        else{
                        fprintf(stderr,"Error: Please enter valid input argument
\n");
```

```
62                              exit(1);
63                              }
64                      }*/
65                      }
66
67              }
68
69                      startup_mode(argv);
70              return 0;
71      }
72
```

```
1  /**
2   * @file unittest.c
3   * @author Justin Yackoski, Deepak Ramadass
4   * @brief This file contains my unit tests.
5   */
6  #include <stddef.h>
7  #include <stdarg.h>
8  #include <setjmp.h>
9  #include <cmocka.h>
10 #include "common.h"
11 #include "3_util.h"
12
13 static char printf_output[MAXLENGTH];
14 struct bought *test; ///< bought test packet
15
16 /**
17  * Wrapper of getaddrinfo
18  * @param node
19  * @param service
20  * @param hints
21  * @param res
22  * @return 0 success 1 failure
23  */
24
25 int __wrap_getaddrinfo(const char *node, const char *service,
26                 const struct addrinfo *hints,
27                 struct addrinfo **res)
28 {
29         *res = malloc(sizeof(struct addrinfo));
30         return (int)mock();
31 }
32
33 /**
34  * Wrapper of sendto
35  * @param sockfd
36  * @param buf
37  * @param len
38  * @param flags
39  * @param dest_addr
40  * @param addrlen
41  * @return actuallen
42  */
43
44 size_t __wrap_sendto(int sockfd, const void *buf, size_t len, int flags,
45         const struct sockaddr *dest_addr, socklen_t addrlen)
46 {
47         size_t actuallen = (size_t)mock();
48         return actuallen;
49 }
50
51 /**
52  * Wrapper of recvfrom
53  * @param sockfd
54  * @param buf
55  * @param len
56  * @param flags
57  * @param src_addr
58  * @param addrlen
59  * @return actuallen
60  */
61
62 ssize_t __wrap_recvfrom(int sockfd, void *buf, size_t len, int flags,
63         struct sockaddr *src_addr, socklen_t *addrlen)
```

```
64  {
65          size_t actuallen = (size_t)mock();
66          //void* answer = mock_ptr_type(void *);
67          memcpy(buf, test, 6);
68          return actuallen;
69  }
70  /**
71   * Wrapper of printf
72   * @param format
73   * @return len
74   */
75
76  int __wrap_printf(const char *format, ...)
77  {
78          va_list(argptr);
79          va_start(argptr, format);
80          int len = vsnprintf(printf_output, MAXLENGTH, format, argptr);
81          if (len >= MAXLENGTH)
82          {
83                  assert_in_range(len, 0, MAXLENGTH-1);
84                  printf_output[MAXLENGTH-1] = 0;
85          }
86          va_end(argptr);
87          return len;
88  }
89  /**
90   * Wrapper of socket
91   * @param domain
92   * @param type
93   * @param protocol
94   * @return sock
95   */
96
97  int __wrap_socket(int domain, int type, int protocol)
98  {
99          int sock = (int)mock();
100         return sock;
101 }
102
103 /*static void prepare_addr_test(void **state)
104 {
105         struct addrinfo *send_addr = NULL;
106         will_return(__wrap_getaddrinfo, 0);
107         assert_int_equal(prepare_addr("::1", "10689", &send_addr), 0);
108         free(send_addr);
109 }
110
111 static void prepare_addr_test_failure(void **state)
112 {
113         struct addrinfo *send_addr = NULL;
114         will_return(__wrap_getaddrinfo, 1);
115         assert_int_equal(prepare_addr("::1", "10689", &send_addr), 1);
116         assert_non_null(send_addr);
117         free(send_addr);
118 }
119 */
120
121 /**
122  * This function tests handle_clients
123  * and checks if its returns a Bought packet
124  * of size 6 bytes
125  */
126 static void handle_clients_test(void **state)
```

```
127 {
128         //struct sockaddr_storage *client_add;
129         struct buy *test_packet;
130         test_packet = malloc(sizeof(struct buy) + 2);
131         test_packet->code = 1;
132         test_packet->seq_num = htonl(10);
133         strncpy(test_packet->name,"GOOG",4);
134         test_packet->quantity_length = 2;
135         char *string = &(test_packet->quantity);
136         string[0] = 'x';
137         string[1] = 'i';
138         //char recvstr[] = "ABC is 3 bytes";
139         //size_t len = sizeof(test_packet) + 1;
140         //will_return(__wrap_recvfrom, len);
141         //will_return(__wrap_recvfrom, (uintptr_t)test_packet);
142         //will_return(__wrap_sendto,6);
143         assert_int_equal(handle_client(test_packet,NULL),6);
144         //assert_string_equal(printf_output,);
145         free(test_packet);
146 }
147
148 /**
149  * This function tests client functionality
150  */
151 static void client_test(void **state)
152 {
153         test = malloc(6);
154         test->code = 2;
155         test->status_code = 1;
156         test->seq_num = htonl(10);
157         char name[4] = "GOOG";
158         char output[] = "XX";
159         uint8_t quanlen = 2;
160         size_t len = 12;
161         will_return(__wrap_getaddrinfo, 0);
162         will_return(__wrap_sendto,len);
163         will_return(__wrap_socket,0);
164         will_return(__wrap_recvfrom,len);
165         //will_return(__wrap_recvfrom, (uintptr_t)test);
166         assert_int_equal((client(name,output,quanlen,"::1","10689")),0);
167         free(test);
168 }
169
170
171
172 /**
173  * This function starts the test program.
174  * @return the exit code of the program
175  */
176 int main(void)
177 {
178         const struct CMUnitTest tests[] =
179         {
180                 //cmocka_unit_test(prepare_addr_test),
181                 //cmocka_unit_test(prepare_addr_test_failure),
182                 //cmocka_unit_test(do_send_recv_test),
183                 cmocka_unit_test(handle_clients_test),
184                 cmocka_unit_test(client_test),
185         };
186
187         return cmocka_run_group_tests(tests, NULL, NULL);
188 }
```

```
1  /**
2   * @file hwk.h
3   * @author Deepak Ramadass, Pooja Burly Prakash, Deepika Rajarajan
4   * @brief This is the header file.
5   */
6  #ifndef HWK_H
7  #define HWK_H
8  #include <stdio.h>
9  #include <stdint.h>
10 #include <string.h>
11
12 /**
13  * @brief This is my structure for storing data
14  * values stores the stock prices
15  * value_names store the stock names
16  * num_values store the number of stocks
17  * __attribute__ ((packed)) ensure struct is packed
18  */
19 struct STOCKS{
20         int64_t* values; ///<the values being stored
21         char ** value_names; ///<the names are stored
22         size_t num_values; ///< how many values there are
23 }__attribute__((packed));///<used to pack
24
25 void store_name(char* name, int line);
26 void store_value(int64_t value, int line);
27 void open_file();
28 void close_file();
29 int startup_mode(char *data[]);
30 int check_arguments(char* line);
31 int initialize(size_t num);
32 void database_cleanup();
33 /**
34   *this pointer provides access to the struct stocks
35 */
36 extern struct STOCKS *database;
37 int transaction_mode(char *ip,char *port);
38 int buy(char stockname[],int num_units);
39 int sell(char *name, int units);
40 int statement();
41 int quote (char stock_name[]);
42 int deposit(int64_t dollars);
43 int withdraw(int64_t dollars);
44 int cmdcaps(char stock[]);
45 int numcheck(char numinput[]);
46 /**
47   * this variable store user balance and stock prices
48 */
49 extern int64_t *user_data;
50 #endif
```

```
CC=gcc
CFLAGS=-ggdb -Wall
LDLIBS=-lcmocka

#EDIT THIS TO BE YOUR GROUP's NAME!
GROUPNAME=hashtag

#EDIT THIS TO BE hwk1, hwk2, hwk3, hwk4, or hwk5
ASSIGNMENT=hwk4

#EDIT THIS TO CONTAIN YOUR .c, .h, Makefile, and .expected files
#DO NOT PUT YOUR COMPILED EXECUTABLES IN THIS LIST
#I WILL JUST DELETE THEM AND RE-COMPILE MYSELF!
FILES=hwk.c main.c unittest.c hwk.h Makefile common.c common.h 3_util.c 3_util.h
 roman.c server_main.c server.c client.c stocknotlisted.txt hashtag_stocknotlist
ed.out.expected hashtag_stocknotlisted.log.expected stockhitszero.txt hashtag_st
ockhitszero.out.expected  hashtag_stockhitszero.log.expected stocknotinputbought
.txt hashtag_stocknotinputbought.out.expected hashtag_stocknotinputbought.log.ex
pected stockbacnotgivenbought.txt hashtag_stockbacnotgivenbought.out.expected ha
shtag_stockbacnotgivenbought.log.expected hashtag_error.txt hashtag_error.out.ex
pected hashtag_error.log.expected

AUTONAME=${GROUPNAME}_${ASSIGNMENT}

all:    hwk server alltest docs

hwk:    hwk.c main.c common.c common.h client.c roman.c

server: server.c server_main.c 3_util.h 3_util.c common.h roman.c

unittest: CFLAGS+=-Wl,--wrap=getaddrinfo,--wrap=sendto,--wrap=printf,--wrap=recvf
rom,--wrap=socket
unittest: 3_util.c 3_util.h common.h roman.c server.c unittest.c common.h common.
c hwk.c client.c roman.c

rununittest: unittest
        #Run unit tests
        ./unittest

alltest: hwk rununittest systemtest1 systemtest2 systemtest3 systemtest4

systemtest1:
        cat stockhitszero.txt
        ./server 10689 & ./hwk ::1 10689 < stockhitszero.txt > out.txt
        killall server
        -diff -u log.txt hashtag_stockhitszero.log.expected
        -diff -u out.txt hashtag_stockhitszero.out.expected
        -rm log.txt
        -rm out.txt

systemtest2:
        cat stocknotlisted.txt
        ./server 10689 & ./hwk ::1 10689 < stocknotlisted.txt >out.txt
        killall server
        -diff -u log.txt hashtag_stocknotlisted.log.expected
        -diff -u out.txt hashtag_stocknotlisted.out.expected
        -rm log.txt
        -rm out.txt

systemtest3:
        cat stocknotinputbought.txt
        ./server 10689 & ./hwk ::1 10689 < stocknotinputbought.txt >out.txt
        killall server
```

```
55          -diff -u log.txt hashtag_stocknotinputbought.log.expected
56          -diff -u out.txt hashtag_stocknotinputbought.out.expected
57          -rm log.txt
58          -rm out.txt
59
60  systemtest4:
61          cat stockbacnotgivenbought.txt
62          ./server 10689 & ./hwk ::1 10689 < stockbacnotgivenbought.txt >out.txt
63          killall server
64          -diff -u log.txt hashtag_stockbacnotgivenbought.log.expected
65          -diff -u out.txt hashtag_stockbacnotgivenbought.out.expected
66          -rm log.txt
67          -rm out.txt
68
69  systemtest5:
70          cat hashtag_error.txt
71          ./hwk ::1 10689 < hashtag_error.txt >out.txt
72          -diff -u log.txt hashtag_error.log.expected
73          -diff -u out.txt hashtag_error.out.expected
74          -rm log.txt
75          -rm out.txt
76  docs:
77          doxygen Doxyfile >/dev/null
78
79  clean:
80          -rm hwk unittest *.out ${AUTONAME}_code.pdf ${AUTONAME}.tar.gz
81          -rm -rf ${GROUPNAME} html latex
82
83  dist: builddist distcheck
84
85  builddist:
86          a2ps -A fill -1 --header="${AUTONAME}" --line-numbers=1 -o ${AUTONAME}_
    code.ps ${FILES}
87          ps2pdf ${AUTONAME}_code.ps
88          -rm -rf ${GROUPNAME}/
89          mkdir ${GROUPNAME}
90          cp ${AUTONAME}_code.pdf ${GROUPNAME}/
91          cp ${GROUPNAME}_${ASSIGNMENT}_report.pdf ${GROUPNAME}/
92          cp ${FILES} ${GROUPNAME}/
93          cp Doxyfile ${GROUPNAME}/
94          tar -cvzf ${AUTONAME}.tar.gz ${GROUPNAME}
95          rm -rf ${GROUPNAME}/
96          rm ${AUTONAME}_code.ps
97
98  distcheck:
99          -rm -rf ${GROUPNAME}/
100         tar -xvzf ${AUTONAME}.tar.gz
101         @echo Checking if a PDF of your code, your main hwk.c file,
102         @echo and your report PDF are included...
103         @test -s ${GROUPNAME}/${AUTONAME}_code.pdf
104         @test -s ${GROUPNAME}/hwk.c
105         @test -s ${GROUPNAME}/${GROUPNAME}_${ASSIGNMENT}_report.pdf
106         @echo Checking if all your expected test results are included...
107         @test `ls *.expected | wc -l` -eq `ls ${GROUPNAME}/*.expected | wc -l` \
108                 || ( ( echo "Some files ending with .expected in the current directory are not in your FI
    LES variable. Move them elsewhere or add them to FILES") && exit 1 )
109         @echo Checking if all your .c files are included...
110         @test `ls *.c | wc -l` -eq `ls ${GROUPNAME}/*.c | wc -l` \
111                 || ( ( echo "Some files ending with .c in the current directory are not in your FILES vari
    able. Move them elsewhere or add them to FILES") && exit 1 )
112         @echo Checking if all your .h files are included...
113         @echo   \(note: if you see \"no such file or directory\" errors immediat
    ely
```

```
	@echo below followed by a passed message, everything is OK\)
	@test `ls *.h | wc -l` -eq `ls ${GROUPNAME}/*.h | wc -l` \
		|| ( ( echo "Some files ending with .h in the current directory are not in your FILES variable. Move them elsewhere or add them to FILES") && exit 1 )
	@test -s ${GROUPNAME}/Makefile
	@echo @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
	@echo Submission sanity checks PASSED!!!
	@echo @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
	@echo THESE FILES LISTED BELOW ARE IN YOUR TARBALL FOR SUBMISSION
	@echo YOU NEED TO DOUBLE CHECK THIS LIST!
	@echo IT IS A GOOD IDEA TO ALSO CHECK THE FILES YOURSELF IN THE
	@echo ${GROUPNAME} DIRECTORY TO ENSURE IT IS CORRECT
	@ls -l ${GROUPNAME}
	@echo @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
	@echo @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
	@echo Now you must submit ${AUTONAME}.tar.gz via Canvas
	@echo Be sure only 1 submission is made by one group member.
	@echo Give a copy of the submission to all group members for
	@echo their records AFTER the designated person has submitted to
	@echo Canvas.  Each group member is responsible for
	@echo the coordination, etc. necessary for on-time submission
	@echo of the group\'s assignment!!
	@echo Email submission is **ONLY** acceptible in the event Canvas
	@echo experienced unexpected downtime.  In that event, email and
	@echo then submit via Canvas when it comes back up.
	@echo @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
	@echo @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```c
#include "common.h"

void handler(int signal)
{
        #ifdef DEBUG
        perror("I got an alarm!\n");
        #endif
}
```

```
1
2  #ifndef COMMON_H
3  #define COMMON_H
4  //#define DEBUG
5
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <sys/socket.h>
11 #include <sys/types.h>
12 #include <netdb.h>
13 #include <unistd.h>
14 #include <assert.h>
15 #include <errno.h>
16 #include <signal.h>
17 #include <arpa/inet.h>
18 #include <stdint.h>
19 #include <ctype.h>
20 #include <time.h>
21 #define MAXLENGTH 2000
22 #define LOG(cmd) fprintf(log_file,"%s ERROR INVALID INPUT\n",strtok(cmd,"\r\n")) //
   /< Logs invalid commands
23 #define LOG_COMM_ERROR fprintf(log_file, "COMMUNICATION ERROR\n");
24 #define LOG_NOSHARES fprintf(log_file, "NO SHARES AVAILABLE FOR PURCHASE\n");
25 FILE *log_file;//points to log file
26 //char input1[MAXLENGTH]={0};
27 //char input2[MAXLENGTH] ={0};
28 //char input3[MAXLENGTH] = {0};
29
30 /**
31   * The BUY structs defiines the contents of the BUY packet
32 */
33 struct buy
34 {
35         uint8_t code;           ///< code set to 1 to indicate BUY packet
36         uint32_t seq_num;       ///< seq_num used for identify packets
37         char name[4];           ///< name corresponds to the STOCK name which ne
   eds to bought
38         uint8_t quantity_length;///< quantity_lenght is the number of characters
    in quantity
39         char quantity;          ///< quantity is the roman numeral of units to b
   e bought
40 } __attribute__ ((packed));     ///< compress the struct
41
42 /**
43   * This BOUGHT struct defines the BOUGHT packet
44 */
45 struct bought
46 {
47         uint8_t code;           ///< code is set to 2 indicating BOUGHT packet
48         uint32_t seq_num;       ///< seq_num is used to identify packets
49         uint8_t status_code;    ///< status_code represents success or failure
50 } __attribute__ ((packed));     ///< compress the struct
51
52 int check_for_invalid_inputs(char *input_string);
53 int handle_client(struct buy *torecv,
54                   struct sockaddr_storage *client_addr);
55 int check_shares(char *name,uint8_t units);
56 void handler(int signal);
57 void open_file();
58 void close_file();
59 int check_arguments(char* line);
```

```
60 uint8_t check_special_case(char *inp);
61 int8_t get_value(char inp);
62 uint8_t roman_to_number(char *src);
63 size_t number_to_roman(char *dst, uint8_t number);
64 uint8_t num_units_quanlen(char *dst);
65 void convert_to_lower(char *input_string);
66 void get_ones_roman(char *store, int digit);
67 void get_tens_roman(char *store, int digit);
68 void get_hundreds_roman(char *store, int digit);
69 int client(char nameinp[4],char output[],uint8_t quanlen,char ip[], char port[])
   ;
70 int no_shares();
71 int communication_error();
72 void client_register_handler();
73 int client_prepare_socket(char ip[],char port[]);
74 #endif
```

```
1  /* utility functions for common things */
2  #include "3_util.h"
3
4  //this function sends a generic data buffer
5  void send_data(void *data, int datalen,
6                  struct sockaddr_storage *client_addr)
7  {
8          socklen_t addr_len = sizeof(struct sockaddr_storage);
9
10         if (client_addr != NULL)
11         {
12                 ssize_t bytes_sent = sendto(sock, data, datalen, 0,
13                         (struct sockaddr *) client_addr, addr_len);
14                 if (bytes_sent < 0)
15                 {
16                         exit_with_error("sendto failed");
17                 }
18         }
19         else
20         {
21                 sprint_hex(data, datalen);
22         }
23 }
24
25 //this function prints a generic data buffer to
26 //the "output" char array (for unit testing)
27 void sprint_hex(uint8_t *data, size_t length)
28 {
29         char myoutput[MAXLENGTH];
30         char tmp[MAXLENGTH];
31         assert(length < 100);
32         myoutput[0] = '\0';
33
34         int i;
35         for (i = 0; i < length; i++)
36         {
37                 sprintf(tmp, "%02x ", data[i]);
38                 strcat(myoutput, tmp);
39                 if (i % 16 == 15)
40                 {
41                         strcat(myoutput, "\n");
42                 }
43         }
44         if (myoutput[strlen(myoutput)-1] != '\n')
45         {
46                 strcat(myoutput, "\n");
47         }
48         //printf("%s\n",myoutput);
49         //output = strdup(myoutput);
50 };
51
52 //This function listens for possible clients
53 //and invokes the handler for each
54 void receive_clients()
55 {
56         while (!stop)
57         {
58                 struct sockaddr_storage client_addr;
59                 int datalen = sizeof(struct buy) + MAXLENGTH;
60                 struct buy *torecv = NULL;
61                 torecv = malloc(datalen);
62                 socklen_t addr_len = sizeof(struct sockaddr_storage);
63
```

```
64                  ssize_t bytes = recvfrom(sock, torecv,
65                                           datalen, 0,
66                                           (struct sockaddr *) &client_addr,
67                                           &addr_len);
68                  if (stop)
69                  {
70                          free(torecv);
71                          break;
72                  }
73
74                  if (bytes < 0)
75                  {
76                          exit_with_error("recvfrom failed");
77                  }
78                  assert(bytes <= 2010);
79
80                  handle_client(torecv, &client_addr);
81                  free(torecv);
82          }
83
84  }
85
86  void handler2(int signal)
87  {
88          stop = 1;
89          if (sock != 0)
90          {
91                  close(sock);
92          }
93  }
94
95  void cleanup()
96  {
97          if (listen_addr)
98                  freeaddrinfo(listen_addr);
99          if (sock)
100                 close(sock);
101 }
102
103 void exit_with_error(char *msg)
104 {
105         perror(msg);
106         cleanup();
107         exit(1);
108 }
109
110 void register_handler()
111 {
112         struct sigaction actinfo;
113         actinfo.sa_handler = handler2;
114         sigfillset(&actinfo.sa_mask); //todo check for error
115         actinfo.sa_flags = 0;
116         sigaction(SIGINT, &actinfo, 0); //todo check for error
117         sigaction(SIGHUP, &actinfo, 0); //todo check for error
118         sigaction(SIGTERM, &actinfo, 0); //todo check for error
119 }
120
121 void prepare_socket(int argc, char *argv[])
122 {
123         struct addrinfo lookup_addr;
124         memset(&lookup_addr, 0, sizeof(struct addrinfo));
125         lookup_addr.ai_family = AF_INET6; //or AF_INET
126         lookup_addr.ai_flags = AI_PASSIVE;
```

```
127         lookup_addr.ai_socktype = SOCK_DGRAM;
128         lookup_addr.ai_protocol = IPPROTO_UDP;
129
130         if (getaddrinfo(NULL, argv[1], &lookup_addr, &listen_addr) != 0)
131         {
132                 exit_with_error("getaddrinfo failed");
133         }
134
135         sock = socket(listen_addr->ai_family, listen_addr->ai_socktype,
136                       listen_addr->ai_protocol);
137         if (sock < 0)
138         {
139                 exit_with_error("socket failed");
140         }
141         if (bind(sock, listen_addr->ai_addr,
142                                 listen_addr->ai_addrlen) < 0)
143         {
144                 exit_with_error("bind failed");
145         }
146
147 }
```

```
1  /* header file for 3_util.c */
2  #include "common.h"
3
4  #ifndef UTIL_H_3
5  #define UTIL_H_3
6
7  extern int stop;
8  extern int sock;
9  extern struct addrinfo *listen_addr;
10 extern char *output;
11
12 //this function needs to be implemented by someone
13 //(it is not in 3_util.c!)
14 int handle_client(struct buy *torecv,
15                   struct sockaddr_storage *client_addr);
16
17 void send_data(void *data, int datalen,
18                   struct sockaddr_storage *client_addr);
19
20 void receive_clients();
21
22 void sprint_hex(uint8_t *data, size_t length);
23
24 void handler2(int signal);
25
26 void cleanup();
27
28 void exit_with_error(char *msg);
29
30 void register_handler();
31
32 void prepare_socket(int argc, char *argv[]);
33
34 #endif
```

```
1  #include "common.h"
2
3  /**
4    * Used to check if invalid roman numerals were entered.
5    * Checks for known cases like "iiii","ixi","ivi","ccvc","ll","vv"
6    * Assumes that the input is all lower case.
7    * This happens when the user is not aware of the roman numeral format
8    * Terminates program if inputs are invalid
9    * @param input_string contains lower case roman numeral that was entered
10   */
11 int check_for_invalid_inputs(char *input_string){
12         convert_to_lower(input_string);
13         int i = 0, v = 0, x = 0, l = 0, c = 0;
14         int pointer = 0;
15         while(input_string[pointer] != '\0'){
16                 if(input_string[pointer] == 'i' && i < 3 && (((input_string[poin
   ter + 1] == 'x' || input_string[pointer + 1] == 'v') && (input_string[pointer +
   2] != 'i')) || input_string[pointer + 1] == 'i' || input_string[pointer+1] == '\0
   '))
17                         i++;
18                 else if(input_string[pointer] == 'v' && v == 0)
19                         v++;
20                 else if(input_string[pointer] == 'x' && x <= 3)
21                         x++;
22                 else if(input_string[pointer] == 'l' && l == 0)
23                         l++;
24                 else if(input_string[pointer] == 'c' && c < 2)
25                         c++;
26                 else
27                         return 1;
28                 pointer++;
29         }
30         return 0;
31 }
32
33 /**
34   * This function is used to identify 4,9,40 and 90,
35   * as they are special cases in Roman Numerals.
36   * @param inp points to two chararcters at a time
37   * of the input entered.
38   * @return the number if the special case exits
39   */
40 uint8_t check_special_case(char *inp){
41         uint8_t number = 0;
42         if(inp[0] == 'i' && inp[1] == 'v')
43                 number =  4;
44         else if(inp[0] == 'i' && inp[1] == 'x')
45                 number =  9;
46         else if(inp[0] == 'x' && inp[1] == 'l')
47                 number =  40;
48         else if(inp[0] == 'x' && inp[1] == 'c')
49                 number =  90;
50         else
51                 number =  0;
52         return number;
53 }
54
55 /**
56   * This function converts the input received to
57   * lower case, thus reducing all comparisons to lower
58   * case alphabets.
59   * @param input_string contains the roman numeral entered.
60   * @return the entire roman numeral in lower case
```

```
 */
void convert_to_lower(char *input_string){
        int i = 0;
        for(i = 0; input_string[i] != '\0'; i++)
                input_string[i] = tolower(input_string[i]);
}

/**
  * This function calculates the value of the roman numeral,
  * the vales are summed up to calculate the integer value of the roman numeral.
  * (except for specials cases handled by check_special_case function).
  * @param inp contains the roman numeral digit whose value needs to calculated.
  * @return the value of the roman numeral digit
 */
int8_t get_value(char inp){
       uint8_t number = 0;
        switch(inp){
                case 'i':
                        number = 1;
                        break;
                case 'v':
                        number = 5;
                        break;
                case 'x':
                        number = 10;
                        break;
                case 'l':
                        number = 50;
                        break;
                case 'c':
                        number = 100;
               default: break;
        }
        return number;
}

/**
 * returns the integer value of the roman number,
 * converts the roman numeral to lower case and checks
 * for special cases(ix,iv,xc,xl) and get its vaule if it exits
 * or gets the value of each roman numeral digit and adds it to "number".
 * Assumes that a vail roman numeral is passed to it.
 * Throws an error if number exceed 255.
 * @param src contains the roman number
 * @return the integer value of the roman number
 */
uint8_t roman_to_number(char *src)
{
        char input_string[1000] = {};
        //char *conversion_pointer;
        //conversion_pointer = input_string;
        char inp[2] = {};
        uint8_t number = 0;
        uint8_t compare = 0;
        strcpy(input_string,src);
        convert_to_lower(input_string);
        //check_for_invalid_inputs(conversion_pointer);
        int i = 0;
        for(i = 0; input_string[i] != '\0'; i++){
                inp[0] = input_string[i];
                if(input_string[i+1] != '\0')
                        inp[1] = input_string[i+1];
                number+= check_special_case(inp);
```

```
124                     if(number != compare)
125                             i++;
126                     else
127                             number+= get_value(input_string[i]);
128                     //if(number < compare)
129                             //parse_error();
130                     compare = number;
131                     }
132             return number;
133
134     }
135
136     /**
137       * This function converts the one’s digit of number to its
138       * corresponding roman numeral representation.
139       * @param store is used to store the output
140       * @param digit contains the one’s digit
141       * @return the roman numeral corresponding to the one’s digit.
142     */
143     void get_ones_roman(char *store, int digit){
144             int i = 0;
145             switch(digit){
146                     case 0: store[0] = ’\0’;
147                     case 1:
148                     case 2:
149                     case 3:
150
151                             for(i = 0;i < digit;i++)
152                                      store[i] = ’i’;
153                             store[digit] = ’\0’;
154                             break;
155                     case 4:
156                             store[0] = ’i’;
157                             store[1] = ’v’;
158                             store[2] = ’\0’;
159                             break;
160                     case 5:
161                     case 6:
162                     case 7:
163                     case 8:
164                             store[0] = ’v’;
165                             for(i = 1;i <=(digit-5);i++)
166                                     store[i] = ’i’;
167                             store[digit-4] = ’\0’;
168                             break;
169                     case 9:
170                             store[0] = ’i’;
171                             store[1] = ’x’;
172                             store[2] = ’\0’;
173                             break;
174                     default :
175                             break;
176             }
177     }
178
179     /**
180       * This function converts the ten’s digit of number to its
181       * corresponding roman numeral representation.
182       * @param store is used to store the output
183       * @param digit contains the ten’s digit
184       * @return the roman numeral corresponding to the ten’s digit.
185     */
186     void get_tens_roman(char *store, int digit){
```

```
187         int i = 0;
188         switch(digit){
189                 case 0: store[0] = '\0';
190                 case 1:
191                 case 2:
192                 case 3:
193                         for(i = 0;i < digit;i++)
194                                 store[i] = 'x';
195                         store[digit] ='\0';
196                         break;
197                 case 4:
198                         store[0] = 'x';
199                         store[1] = 'l';
200                         store[2] = '\0';
201                         break;
202                 case 5:
203                 case 6:
204                 case 7:
205                 case 8:
206                         store[0] = 'l';
207                         for(i = 1;i <= (digit-5);i++)
208                                 store[i] = 'x';
209                         store[digit-4] = '\0';
210                         break;
211                 case 9:
212                         store[0] = 'x';
213                         store[1] = 'c';
214                         store[2] = '\0';
215                         break;
216         }
217 }
218
219 /**
220   * This function converts the hundred's digit of number to its
221   * corresponding roman numeral representation. It assumes that
222   * 2 is the greatest hundreth digit that will exist.
223   * @param store is used to store the output
224   * @param digit contains the hundred's digit
225   * @return the roman numeral corresponding to the hundred's digit.
226 */
227  void get_hundreds_roman(char *store, int digit){
228         int i = 0;
229         switch(digit){
230                 case 0: store[0] = '\0';
231                 case 1:
232                 case 2:
233                         for(i = 0;i < digit; i++)
234                                 store[i] = 'c';
235                         store[digit] = '\0';
236                 default:break;
237         }
238 }
239
240
241
242 /**
243  * called to change the number back to a roman numeral
244  * it seperates the number into one's,tens's and hundred's
245  * digit, calls the corresponding functions to get their
246  * roman numeral representations.
247  * @param dst stores the roman numeral that is obtained
248  * @param number to be coverted
249  * @return the size of the roman numeral obtained.
```

```
 */
size_t number_to_roman(char *dst, uint8_t number)
{
        char store[sizeof(dst)] = {};
        uint8_t ones_digit = number % 10;
        uint8_t tens_digit = (number/10) % 10;
        uint8_t hundreds_digit = number / 100;
        dst[0] = '\0';
        get_hundreds_roman(store, hundreds_digit);
        strncat(dst,store,strlen(store));
        get_tens_roman(store, tens_digit);
        strncat(dst,store,strlen(store));
        get_ones_roman(store, ones_digit);
        strncat(dst,store,strlen(store));
        return strlen(dst)+1;
}
```

```c
/**
 * @file server_main.c
 * @author Deepak Ramadass
 * @brief This file contains my main.
 */
#include "common.h"
#include "3_util.h"

/**
 * This is the main function that starts the program.
 * @param argc the number of arguments
 * @param argv the command line arguments
 *   (the 0th argument is the program name)
 * @return  0 the exit code of the program
 */
int main(int argc, char *argv[])
{
        if (argc > 2)
        {
                exit(1);
        }
        int i = 0;
        for(i = 0; i <strlen(argv[1]); i++){
                if(argv[1][i] >= '0' && argv[1][i] <='9')
                        continue;
                else
                        exit(1);
        }
        register_handler();
        prepare_socket(argc, argv);
        receive_clients();
        cleanup();

        return 0;
}
```

```
/** server with reusable logic, better organization,
 * and unit tests using 3_util.c logic
 * @ author Deepak Ramadass
 */

#include "common.h"
#include "3_util.h"

int stop = 0;
int sock = 0;
int GOOG = 200;
int BAC = 200;
struct addrinfo *listen_addr = NULL;
char *output = NULL;
uint8_t my_responses[100] = {0};
uint8_t seq_num_store[100] = {0};

/**
  * This function checks the number of share corresponding to GOOG
  * and BAC are available and reduces the global value by the no. of
  * units.
  * @param name - the name of the stock, units - the no. of shares
  * @return 0/1 - success/failure
*/
int check_shares(char *name,uint8_t units){
        if (strncmp(name,"GOOG",4) == 0)
        {
                if(GOOG >= units){
                        GOOG-= units;
                        return 0;
                }
                else
                        return 1;
        }
        else if (strncmp(name,"BAC",3) == 0)
        {
                if(BAC >= units){
                        BAC-= units;
                        return 0;
                }
                else
                        return 1;
        }
        else
        {
                return 1;
        }


}

/**
  * This function processes the data received by a single client,
  * calls the check_shares function, constructs the BOUGHT packect,
  * and resends BOUGHT packets if the sequence numbers were seen before
  * @param torec points to the BUY packet
  * @param client_addr points to the address to send the BOUGHT packet
  * @return returns the size of the BOUGHT packet for unittesting
*/
int handle_client(struct buy *torecv,
                struct sockaddr_storage *client_addr)
{
        if(torecv->code != 1)
```

```
64                      return 1;
65              printf("NEW BUY PACKET:\n");
66              printf("seq_num: %u\n", ntohl(torecv->seq_num));
67
68              char name[5] = {'\0'};
69              strncpy(name,torecv->name,4);
70              printf("name: %s\n", name);
71
72              uint8_t length = torecv->quantity_length;
73              char *roman_numeral = malloc(length +1);
74              memcpy(roman_numeral,&(torecv->quantity),length);
75              roman_numeral[length] = '\0';
76
77              struct bought tosend;
78              tosend.code = 2;
79
80              if(check_for_invalid_inputs(roman_numeral) == 1){
81                      tosend.status_code = 2;
82                      goto label1;
83              }
84              if(my_responses[ntohl(torecv->seq_num)] != 0){
85                      printf("resending packect with seq_num %u\n",ntohl(torecv->seq_num));
86                      tosend.seq_num = htonl(torecv->seq_num);
87                      tosend.status_code = my_responses[ntohl(torecv->seq_num)];
88              }
89
90              else{
91                      uint8_t units = roman_to_number(roman_numeral);
92                      printf("QUANTITY REQUESTED %u\n",units);
93                      printf("STOCKS AVAILABLE GOOG: %d  BAC: %d \n",GOOG,BAC);
94
95                      if(check_shares(name, units) == 0)
96                              tosend.status_code = 1;
97                      else
98                              tosend.status_code = 2;
99
100             label1: tosend.seq_num = htonl(torecv->seq_num);
101                     my_responses[ntohl(torecv->seq_num)] = tosend.status_code;
102             }
103
104             send_data(&tosend, sizeof(struct bought), client_addr);
105             free(roman_numeral);
106             return sizeof(tosend);
107     }
108
109     //run with port # as the first argument
110     //talk to using netcat, e.g.: nc -u ::1 10689
```

```
/**
 * @file client.c
 * @author Justin Yackoski, Deepika Rajarajan
 * @brief Client with reused logic from 1_clientrexmit.c
 */

#include "common.h"
//run with ip and port as arguments
//talk to using netcat e.g:
//nc -u -l -k -w 0 ::1 10689
/**
 * client program
 * @param nameinp name of the stock
 * @param output units in roman numeral
 * @param quanlen number of characters of roman numeral
 * @param ip ip used in transfer
 * @param port 10689 used in transfer
 * @return 1 failure 0 success
 */

int client(char nameinp[4],char output[],uint8_t quanlen,char ip[],char port[])
{
        client_register_handler();

        struct addrinfo lookup_addr;
        memset(&lookup_addr, 0, sizeof(struct addrinfo));
        lookup_addr.ai_family = AF_UNSPEC;
        lookup_addr.ai_socktype = SOCK_DGRAM;
        lookup_addr.ai_protocol = IPPROTO_UDP;
        struct addrinfo *send_addr;
        if (getaddrinfo(ip,port, &lookup_addr, &send_addr)!=0)
        {
        #ifdef DEBUG
        perror("getaddrinfo failed");
        #endif
        communication_error();
        LOG_COMM_ERROR;
        fflush(log_file);
        freeaddrinfo(send_addr);
        return 1;
        }
        int sock = socket(send_addr->ai_family,send_addr->ai_socktype,send_addr->ai_protocol);
        if(sock<0)
        {
        #ifdef DEBUG
        perror("socket failed");
        #endif
        communication_error();
        LOG_COMM_ERROR;
        fflush(log_file);
        freeaddrinfo(send_addr);
        return 1;
        }

        struct bought torecv;
        int datalen = sizeof(struct buy) + (quanlen -1);
        struct buy *tosend = malloc(datalen);
        tosend->code = 1;
        tosend->seq_num = htonl(random() %254 +1);
//Stock name is no null terminator
        char *strname = tosend->name;
        int j;
```

```
63          for(j=0;j<4;j++)
64          {
65                          strname[j] = nameinp[j];
66          }
67          tosend->quantity_length = quanlen;
68  //quanity_length with no null terminator
69          char *strquan = &(tosend->quantity);
70          int i = 0;
71          for(i=0;i<quanlen;i++)
72          {
73                          strquan[i] = output[i];
74          }
75          ssize_t bytes_sent = sendto(sock, tosend, datalen, 0,
76                          send_addr->ai_addr, send_addr->ai_addrlen);
77          if(bytes_sent !=datalen)
78          {
79          #ifdef DEBUG
80          perror("sendto failed");
81          #endif
82          communication_error();
83          LOG_COMM_ERROR;
84          fflush(log_file);
85          freeaddrinfo(send_addr);
86          free(tosend);
87          close(sock);
88          return 1;
89          }
90
91  //Set alarm for 2 secs
92          alarm(2);
93
94          socklen_t addr_len = send_addr->ai_addrlen;
95          ssize_t bytes = recvfrom(sock, &torecv, sizeof(struct bought), 0,send_ad
    dr->ai_addr, &addr_len);
96          if(errno != EINTR)
97          {
98                  if(bytes < 0)
99                  {
100                         #ifdef DEBUG
101                         perror("recvfrom failed");
102                         #endif
103                         communication_error();
104                         LOG_COMM_ERROR;
105                         fflush(log_file);
106                         freeaddrinfo(send_addr);
107                         free(tosend);
108                         close(sock);
109                         return 1;
110                 }
111         }
112
113         else
114         {
115                 errno = 0;
116                 int timer;
117 //Resend the packet to 5 times
118                 for(timer = 0; timer <5;timer++)
119                 {
120                 alarm(2);
121                 bytes_sent = sendto(sock, tosend, datalen, 0, send_addr->ai_addr
    , send_addr->ai_addrlen);
122                 bytes = recvfrom(sock, &torecv, sizeof(struct bought), 0, send_a
    ddr->ai_addr, &addr_len);
```

```
123                             if(errno != EINTR)
124                             {
125                                     if(bytes<0)
126                                     {
127                                             #ifdef DEBUG
128                                             perror("recvfrom failed\n");
129                                             #endif
130                                             communication_error();
131                                             LOG_COMM_ERROR;
132                                             fflush(log_file);
133                                             freeaddrinfo(send_addr);
134                                             free(tosend);
135                                             close(sock);
136                                             return 1;
137                                     }
138                                     else if(bytes>0)
139                                     {
140                                             goto label1;
141                                     }
142                             }
143                             else
144                             {
145                             errno = 0;
146                             }
147                     }
148
149                     if(timer>=5)
150                     {
151                     alarm(0);
152                     communication_error();
153                     LOG_COMM_ERROR;
154                     fflush(log_file);
155                     freeaddrinfo(send_addr);
156                     free(tosend);
157                     close(sock);
158                     return 1;
159
160                     }
161             }
162             if(bytes > 0)
163             {
164     //if bytes are received then turn off the alarm
165             label1: alarm(0);
166             }
167             else
168             {
169             communication_error();
170             LOG_COMM_ERROR;
171             fflush(log_file);
172             freeaddrinfo(send_addr);
173             free(tosend);
174             close(sock);
175             return 1;
176             }
177             if(torecv.status_code == 2)
178             {
179             no_shares();
180             LOG_NOSHARES;
181             fflush(log_file);
182             freeaddrinfo(send_addr);
183             free(tosend);
184             close(sock);
185             return 1;
```

```
186         }
187         else if (torecv.status_code == 1)
188         {
189         }
190         else
191         {
192         freeaddrinfo(send_addr);
193         free(tosend);
194         close(sock);
195         return 1;
196         }
197         freeaddrinfo(send_addr);
198         free(tosend);
199         close(sock);
200 return 0;
201 }
```

```
1 STOCKS 1
2 STOCK ABC $100
3 DEPOSIT $10000
4 BUY ABC 10
```

```
1 STARTUP MODE READY
2 TRANSACTION MODE READY
3 $10000 DEPOSITED ~~~ BALANCE NOW $10000
4 NO SHARES AVAILABLE FOR PURCHASE
```

hashtag_stocknotlisted.log.expected


```
1  STOCKS 1 BALANCE $0 FEE $0 - - - -
2  STOCK ABC $100 BALANCE $0 FEE $0 - - - -
3  DEPOSIT $10000 BALANCE $10000 FEE $0 - - - -
4  NO SHARES AVAILABLE FOR PURCHASE
```

```
1 STOCKS 2
2 STOCK GOOG $1
3 STOCK BAC $2
4 DEPOSIT $10000
5 BUY GOOG 1
6 BUY GOOG 1
7 BUY BAC 1
8 BUY BAC 1
9 BUY BAC 1
```

```
STARTUP MODE READY
TRANSACTION MODE READY
$10000 DEPOSITED ~~~ BALANCE NOW $10000
1 SHARES OF GOOG BOUGHT FOR $1 TOTAL ~~~ BALANCE NOW $9999
ERROR INVALID INPUT
1 SHARES OF BAC BOUGHT FOR $2 TOTAL ~~~ BALANCE NOW $9989
1 SHARES OF BAC BOUGHT FOR $1 TOTAL ~~~ BALANCE NOW $9980
ERROR INVALID INPUT
```

```
STOCKS 2 BALANCE $0 FEE $0 - - - -
STOCK GOOG $1 BALANCE $0 FEE $0 - - - -
STOCK BAC $2 BALANCE $0 FEE $0 - - - -
DEPOSIT $10000 BALANCE $10000 FEE $0 - - - -
BUY GOOG 1 BALANCE $9999 FEE $0 GOOG 1 $1 $0
BUY GOOG 1 ERROR INVALID INPUT
BUY BAC 1 BALANCE $9989 FEE $8 BAC 1 $2 $1
BUY BAC 1 BALANCE $9980 FEE $8 BAC 2 $1 $0
BUY BAC 1 ERROR INVALID INPUT
```

```
1 STOCKS 1
2 STOCK GOOG $10
3 DEPOSIT $10000
4 BUY ABC 20
```

```
1 STARTUP MODE READY
2 TRANSACTION MODE READY
3 $10000 DEPOSITED ~~~ BALANCE NOW $10000
4 ERROR INVALID INPUT
```

```
1 STOCKS 1 BALANCE $0 FEE $0 - - - -
2 STOCK GOOG $10 BALANCE $0 FEE $0 - - - -
3 DEPOSIT $10000 BALANCE $10000 FEE $0 - - - -
4 BUY ABC 20 ERROR INVALID INPUT
```

```
STOCKS 1
STOCK GOOG $100
DEPOSIT $10000
BUY BAC 10
```

```
1 STARTUP MODE READY
2 TRANSACTION MODE READY
3 $10000 DEPOSITED ~~~ BALANCE NOW $10000
4 ERROR INVALID INPUT
```

```
1 STOCKS 1 BALANCE $0 FEE $0 - - - -
2 STOCK GOOG $100 BALANCE $0 FEE $0 - - - -
3 DEPOSIT $10000 BALANCE $10000 FEE $0 - - - -
4 BUY BAC 10 ERROR INVALID INPUT
```

```
1 STOCKS 1
2 STOCK GOOG $10
3 DEPOSIT $1000
4 BUY GOOG 30
```

hashtag_error.out.expected


```
1  STARTUP MODE READY
2  TRANSACTION MODE READY
3  $1000 DEPOSITED ~~~ BALANCE NOW $1000
4  COMMUNICATION ERROR
```

```
1 STOCKS 1 BALANCE $0 FEE $0 - - - -
2 STOCK GOOG $10 BALANCE $0 FEE $0 - - - -
3 DEPOSIT $1000 BALANCE $1000 FEE $0 - - - -
4 COMMUNICATION ERROR
```