```c
/* Bobo Shi */
/*
type 'gcc shellA.c -o shA' to compile
type './shA' to run program.
Test cases are in testA.txt file.
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>

#define MAX_LINE 80 /* 80 chars per line, per command, should be enough. */
#define HIS_SIZE 5

/**
 * setup() reads in the next command line, separating it into distinct
     tokens
 * using whitespace as delimiters. setup() sets the args parameter as a
 * null-terminated string.
 */

void setup(char inputBuffer[], char *args[],int *background)
{
    int length, /* # of characters in the command line */
        i,      /* loop index for accessing inputBuffer array */
        start,  /* index where beginning of next command parameter is */
        ct;     /* index of where to place the next parameter into args[] */

    ct = 0;

    /* read what the user enters on the command line */
    length = read(STDIN_FILENO, inputBuffer, MAX_LINE);

    start = -1;
    if (length == 0)
        exit(0);            /* ^d was entered, end of user command stream */
    if (length < 0){
        perror("error reading the command");
    exit(-1);               /* terminate with error code of -1 */
    }

    /* examine every character in the inputBuffer */
    for (i = 0; i < length; i++) {
        switch (inputBuffer[i]){
        case ' ':
        case '\t' :                 /* argument separators */
            if(start != -1){
                args[ct] = &inputBuffer[start];   /* set up pointer */
                ct++;
            }
            inputBuffer[i] = '\0'; /* add a null char; make a C string */
            start = -1;
            break;

        case '\n':                      /* should be the final char examined */
```

```
                if (start != -1){
                    args[ct] = &inputBuffer[start];
                    ct++;
                }
                inputBuffer[i] = '\0';
                args[ct] = NULL; /* no more arguments to this command */
                break;

            case '&':
                *background = 1;
                inputBuffer[i] = '\0';
                break;

            default :                   /* some other character */
                if (start == -1)
                    start = i;
    }
    }
    args[ct] = NULL; /* just in case the input line was > 80 */
}

/*void add_command_to_history( const char *command )
{

}*/

struct history{
  char buffer[MAX_LINE];
  char *args[MAX_LINE/2+1];
  int count;
  int background;
};

int main(void)
{
    char inputBuffer[MAX_LINE]; /* buffer to hold the command entered */
    int background;             /* equals 1 if a command is followed by '&'
        */
    char *args[MAX_LINE/2+1];/* command line (of 80) has max of 40 arguments
        */
    int i,j,rr_flag; //bobo add
    pid_t pid;
    int status;
    char *runargs[MAX_LINE/2+1];
    int run_bg;
    struct history his[HIS_SIZE+1];
    int command_count=0;
    char temp[MAX_LINE];
    int rr_num, rr_i;

    //printf("sizeof= %d", sizeof(his[0].args));
    for(i=0; i<HIS_SIZE+1; i++){
      for(j=0; j<MAX_LINE/2+1; j++){
//  printf("%d %d\n", i, j);
    his[i].args[j]=NULL;
      }
```

```c
        his[i].count=0;
    }


    while (1){                  /* Program terminates normally inside setup */

      for(i=0; i<MAX_LINE/2+1; i++){
    runargs[i]=NULL;
      }
      run_bg=0;

      background = 0;
      printf("SystemsIIShell->");
      fflush(0);
      setup(inputBuffer, args, &background);       /* get next command */
      if(args[0]==NULL) continue;
      // if it is 'rr' or 'r num'
    if (strcmp(args[0], "rr") == 0){
      if (his[HIS_SIZE].count<1) {
        printf("No recent command\n");
        break;
      }
      else{
        j=0;
        while(his[HIS_SIZE].args[j]){
           if(his[HIS_SIZE].args[j]){
        strcpy(temp,his[HIS_SIZE].args[j]);
        runargs[j]=malloc(sizeof(temp));
        strcpy(runargs[j],temp);
        printf("%s ",runargs[j]);
           }
           j++;
        }
        printf("\n");

        run_bg=his[HIS_SIZE].background;

      }
    }

    else if (strcmp(args[0], "r")==0){
      //rr_num = atoi(args[1]);
      //if(rr_num = atoi(args[1]))
      if(args[1]){
        if(rr_num= atoi(args[1])){
          rr_flag=0;
          for (i=0; i<HIS_SIZE+1; i++){
        if(his[i].count==rr_num){
          j=0;
          while(his[i].args[j]){
            if(his[i].args[j]){
              strcpy(temp,his[i].args[j]);
              runargs[j]=malloc(sizeof(temp));
              strcpy(runargs[j],temp);
              //            runargs[j]=his[i].args[j];
              printf("%s ",runargs[j]);
```

```c
            }
            j++;
          }
          printf("\n");

          run_bg=his[i].background;
          rr_flag=1;
          break;
      }
        }
        if (rr_flag==0){
    printf("the num you indicate is not in history\n");
    j=0;
    while(args[j]){
      if(args[j])
        runargs[j]=args[j];
      j++;
    }
    run_bg=background;
        }
      }
      else {
        printf("%s is not a num!\n", args[1]);
        j=0;
        while(args[j]){
    if(args[j])
      runargs[j]=args[j];
    j++;
        }
        run_bg=background;

      }
    }
    else{
      printf("no num!\n");
      j=0;
      while(args[j]){
        if(args[j])
    runargs[j]=args[j];
        j++;
      }
      run_bg=background;
    }
  }
    else{
      j=0;
      while(args[j]){
        if(args[j])
    runargs[j]=args[j];
        j++;
      }
      run_bg=background;
    }

  if(strcmp(runargs[0],"h") == 0 || strcmp(runargs[0],"history")==0){
    if (command_count<5)
```

```c
      for (i=HIS_SIZE-command_count+1; i<HIS_SIZE+1; i++){
        if ( his[i].count > 0 && his[i].buffer != NULL && his[i].args[0] !
              = NULL){
      printf("%7d  ", his[i].count);
      j=0;
      while(his[i].args[j]){
        if(his[i].args[j])
          printf(" %s", his[i].args[j]);
        j++;
      }
      if(his[i].background==1)
        printf(" &");
      printf("\n");
        }
      }
    else{
      for (i=1; i<HIS_SIZE+1; i++)
        {
      printf("%7d  ", his[i].count);
      j=0;
      while(his[i].args[j]){
        if(his[i].args[j]){
          printf(" %s", his[i].args[j]);
        }
        j++;
      }
      if(his[i].background==1)
        printf(" &");
      printf("\n");
        }
    }

  }


  else{
    pid = fork(); /* fork a child */

    if (pid < 0){ /* error occurred */
      fprintf(stderr, "Fork Failed\n");
      return 1;
    }
    else if (pid == 0){ /* child process*/

      if (execvp (*runargs, runargs) < 0){ /* if wrong command */
        printf("*** ERROR: exec failed\n");
        exit(1);
      }
    }
    else { /* parent process */
      if (run_bg==0) /* wait if '&' */
        while (wait(&status) != pid);
    }

  }
    // shift value in his one by one
```

```c
        command_count++;
        for (i=0; i<HIS_SIZE; i++){
          if(command_count+i > HIS_SIZE+1){
            strcpy(his[i].buffer,"");
            j=0;
            while(his[i].args[j]){
               if(his[i].args[j]){
            free(his[i].args[j]);
            his[i].args[j]=NULL;
               }
               j++;
            }
            his[i].background=0;
          }
          if(command_count+i > HIS_SIZE){
            j=0;
            while(his[i+1].args[j]){
               if(his[i+1].args[j]){
            strcpy(temp,his[i+1].args[j]);
            his[i].args[j] = malloc(sizeof(temp));
            strcpy(his[i].args[j],temp);
               }
               j++;
            }
            strcpy(his[i].buffer,his[i+1].buffer);
            his[i].background=his[i+1].background;
            his[i].count=his[i+1].count;
          }
        }

        //store command to his[HIS_SIZE]
        if(command_count>1){
          i=0;
          while(his[HIS_SIZE].args[i]){
            //printf("free last: his[HIS_SIZE].args[%d]=%s
                \n",i,his[HIS_SIZE].args[i]);
            if(his[HIS_SIZE].args[i]){
              free(his[HIS_SIZE].args[i]);
              his[HIS_SIZE].args[i]=NULL;
            }
            i++;
          }
          his[HIS_SIZE].background=0;
          strcpy(his[HIS_SIZE].buffer,"");
        }

        i=0;
        while(runargs[i]){
          if(runargs[i]){
            strcpy(temp,runargs[i]);
            his[HIS_SIZE].args[i]=malloc(sizeof(temp));
            strcpy(his[HIS_SIZE].args[i],temp);
          }
          i++;
        }
        his[HIS_SIZE].count=command_count;
```

```
        his[HIS_SIZE].background=run_bg;

        i=0;
        while(runargs[i]){
          if(runargs[i]){
            // free(runargs[i]);
            runargs[i]=NULL;
          }
          i++;
        }
        run_bg=0;
        }
    }
```