```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include <unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#define MAX_WORD 10
#define MAX_CHAR 100
#define DEL " "
int input_redirection_flag;
int output_redirection_flag;
int piping_flag;
char* input_file= NULL;
char* output_file= NULL;
//////////////////////////////////////
void remove_endofline(char line[])
{
int i=0;
while(line[i]!='\n')
i++;

line[i]='\0';
}

////////////////////////////
void read_line(char line[])
{
char* ret =fgets(line, MAX_CHAR,stdin);
```

```
//printf("%s\n",line);
remove_endofline(line);
if(strcmp(line,"exit")==0||ret==NULL)
exit(0);
}
////////////////////////////////////////
```

```
int process_line(char* temp[],char line[])
{
int i =0;
temp[i]=strtok(line,DEL);

while(temp[i]!= NULL){
i++;
temp[i]= strtok(NULL,DEL);

}
return 1;

}
////////////////////////////////////////////////////////
```

```
int pipe_and_redirection_check(char* temp[]){
int i=0;
while(temp[i]!= NULL)
```

```
{
if(strcmp(temp[i],">")==0){
output_redirection_flag=1;
output_file = temp[i+1];
return i;
}
else if(strcmp(temp[i],"<")==0){
input_redirection_flag =1;
input_file = temp[i+1];
return i;
}
else if(strcmp(temp[i],"|")==0){
 piping_flag=1;

return i;
}
i++;
}
return i;
}

//////////////////////////////////////////////////
void check_line(char* temp[])
{
int i =0;
int pipe_count=0;
int output_redirection_count =0;
int input_redirection_count =0;
int total_count;
```

```
if(temp[i]== NULL)
{
printf("no command\n");
return ;
}
while(temp[i]!=NULL)
{
if(strcmp(temp[i],">")==0)
output_redirection_count ++;

else if(strcmp(temp[i],"<")==0)
input_redirection_count++;

else if(strcmp(temp[i],"|")==0)
pipe_count++;

i++;
}
total_count=input_redirection_count+output_redirection_count+pipe_count;
if(total_count > 1){
printf("SORRY MY SHELL DOESN'T HANDLE THIS CASE\n");
temp[0]=NULL;

}



}

////////////////////////////////////////////////////////////////////////
```

```
int read_parse_line(char* args[],char* piping_args[],char line[])
{
char* temp[MAX_WORD];
int i=0;
int pos;
read_line(line);
process_line(temp,line);
check_line(temp);
pos =pipe_and_redirection_check(temp);
while(i<pos){
args[i]=temp[i];
i++;
}
args[i]=NULL;
if(piping_flag==1)
{
int y=0;
while(temp[i]!=NULL)
{
piping_args[y]=temp[i];
i++;
y++;
}
}
return 1;
}
```

//////////////////////////////////////

```
/*void piping_handle( char* args[] ,char* piping_args[],int pipefd[])
{
int pid;
int i;
pid=fork();

if(pid==0)
{

dup2(pipefd[1],1);
 close(pipefd[0]);
 close(pipefd[1]);

execvp(args[0],args);
error("failed to exec command 1");
}
else
{
dup2(pipefd[0],0);
close(pipefd[1]);
close(pipefd[0]);
execvp(piping_args[0],piping_args);
error("failed to exec command 2");
}

}
*/
//////////////////////////////////////////////////////////
```

```
int main()
{
   char* args[MAX_WORD];
   char line[MAX_CHAR];
   char* piping_args[MAX_WORD];
   int pipefd[2];
   pipe(pipefd);

   while(read_parse_line(args,piping_args,line))
   {
   pid_t pid = fork();

   if(pid==0)
   {
   ////////////////////////////////////////////////
    if(input_redirection_flag== 1&& input_file!=NULL)
    dup2(open(input_file,O_RDWR|O_CREAT,0777),0);
    ////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////
    else if(output_redirection_flag== 1&& output_file!=NULL)
    dup2(open(output_file,O_RDWR|O_CREAT,0777),1);
    ////////////////////////////////////////////////////////
    else if(piping_flag==1);
    {
    //iping_handle(args,piping_args,pipefd);
    //exit(0);
    }
    execvp(args[0],args);
```

```
    }

    else
    {
     waitpid(pid, 0);
     input_redirection_flag=0;
     output_redirection_flag=0;
     output_file= NULL;
     input_file =NULL;
     piping_flag =0;


    }

    }

    return 0;
}
```