Read me file

This Shell is created to implement ;(semi-colon operator) , The command before the semi-colon is executed first then the next one from left to right, and & (And operator) The command before the semi-colon is executed both left and right command parallelly.

1:- I have added data structures for ;(Sqlcmd) and & (Paracmd)

```
struct Sqlcmd {
  int type;          // ;
  struct cmd *left;  // left side of seq
  struct cmd *right; // right side of seq
};

struct Paracmd {
  int type;          // ;
  struct cmd *left;  // left side of seq
  struct cmd *right; // right side of seq
};



struct cmd*
Sqlcmd(struct cmd *left, struct cmd *right)
{
  struct Sqlcmd *cmd;

  cmd = malloc(sizeof(*cmd));
  memset(cmd, 0, sizeof(*cmd));
  cmd->type = ';';
  cmd->left = left;
  cmd->right = right;
  return (struct cmd*)cmd;
}
struct cmd*
Paracmd(struct cmd *left, struct cmd *right)
{
  struct Paracmd *cmd;

  cmd = malloc(sizeof(*cmd));
  memset(cmd, 0, sizeof(*cmd));
  cmd->type = '&';
  cmd->left = left;
  cmd->right = right;
  return (struct cmd*)cmd;
}
```

1) In the runcmd function I added case for ‘ ; ‘ and ‘&’ which executes the code for semi-colon case ';' and ‘&’ case:

```
  case ';': //description for ;
```

```
    scmd = (struct Sqlcmd*)cmd;
    if(fork1() == 0 )

      runcmd(scmd->left);
      wait(NULL);

    runcmd(scmd->right);
    break;

  case '&':   //description for &
    pacmd = (struct Paracmd*)cmd;
    if(fork1() == 0)

      runcmd(pacmd->left);
      runcmd(pacmd->right);
    break;
  }
  exit(0);
   }
```

2) In the data structure struct cmd* parseexec(char **ps, char *es) I modified a particular line which will parse ‘ ; ‘ and ‘&’ character.
   while(!peek(ps, es, ";")) //replaced ‘ | ‘ with ‘ ; ‘ to parse semi-colon

```
struct cmd*
parseline(char **ps, char *es)
{
  struct cmd *cmd;
  cmd = parseexec(ps, es);
  while(peek(ps, es, "&")){
    gettoken(ps, es, 0, 0);
   // fprintf(stderr, "check\n");
    cmd = Paracmd(cmd, parseline(ps, es));
  }
  if(peek(ps, es, ";")){ //fprintf(stderr, "check1\n");
    gettoken(ps, es, 0, 0);
    cmd = Sqlcmd(cmd, parseline(ps, es));
  }

  return cmd;
}

struct cmd*
parsepipe(char **ps, char *es)
{
  struct cmd *cmd;

  cmd = parseexec(ps, es);
  if(peek(ps, es, ":&")){
    gettoken(ps, es, 0, 0);
    cmd = pipecmd(cmd, parsepipe(ps, es));
  }
  return cmd;}
```

Test Cases:-

First we have ran the file over gcc complier in UNIC platform:-

```
chaets@DESKTOP-54H15EU:~/OS> gcc -o T6 T6.c
chaets@DESKTOP-54H15EU:~/OS> ./T6
```

1:- $ touch temp.txt; ls -l temp.txt

```
$ touch temp.txt; ls -l temp.txt
-rw-rw-rw- 1 chaets users 13 Sep 16 19:44 temp.txt
```

2:- $ pwd & cat temp.txt

```
$ pwd & cat temp.txt
/home/chaets/OS
This is cool
```

3:- $ ls & sleep 5

```
$ ls & sleep 5
T1  T1.c  T2  T2.c  T3  T3.c  T4  T4.c  T5  T5.c  T6  T6.c  temp.txt
```

4:- $ ls ; cat temp.txt & sleep 5

```
$ ls ; cat temp.txt & sleep 5
T1  T1.c  T2  T2.c  T3  T3.c  T4  T4.c  T5  T5.c  T6  T6.c  temp.txt
This is cool
```

5:- $ echo "time to sleep"; echo "wake up" ; sleep 10 & echo "sleeping..."

```
$ echo "time to sleep"; echo "wake up" ; sleep 10 & echo "sleeping..."
"time to sleep"
"wake up"
"sleeping..."
```

6:- ls ; cat temp.txt & sleep 5

```
$ ls ; cat temp.txt & sleep 5
T1  T1.c  T2  T2.c  T3  T3.c  T4  T4.c  T5  T5.c  T6  T6.c  temp.txt
This is cool
```

```
$ touch temp.txt; ls -l temp.txt
-rw-rw-rw- 1 chaets users 13 Sep 16 19:44 temp.txt
$ pwd & cat temp.txt
/home/chaets/OS
This is cool
$ pwd ; cat temp.txt
/home/chaets/OS
This is cool
$ ls & sleep 5
T1  T1.c  T2  T2.c  T3  T3.c  T4  T4.c  T5  T5.c  T6  T6.c  temp.txt
$ ls ; cat temp.txt & sleep 5
T1  T1.c  T2  T2.c  T3  T3.c  T4  T4.c  T5  T5.c  T6  T6.c  temp.txt
This is cool
$ echo "time to sleep"; echo "wake up" ; sleep 10 & echo "sleeping..."
"time to sleep"
"wake up"
"sleeping..."
```