# Lezione 15

Completare il programma `smallsh`, definendo il codice delle funzioni `gettok` e `runcommand`.

```
int gettok(char **outptr)
{
  int type;
  *outptr=tok;

  while(*ptr==' ' || *ptr=='\t')
    ptr++;

  switch(*ptr) {
  case '\n':
    type=EOL;
    ptr++;
    break;
  case '&':
    type=AMPERSAND;
    ptr++;
    break;
  case ';':
    type=SEMICOLON;
    ptr++;
    break;
  default:
    type=ARG;

    while(inarg(*ptr))
      *tok++=*ptr++;

  }

  *tok++='\0';
  return type;
}

int runcommand(char **cline, int where)
{
  pid_t pid;
  int status;

  switch(pid=fork()) {
  case -1:
    perror("Errore nella chiamata alla fork");
    return -1;
  case 0:
    execvp(*cline,cline);
    perror(*cline);
```

```
    exit(1);
  default:

    if(where==BACKGROUND)
      {
        printf("Process id: %d\n",pid);
        return 0;
      }
    else
      {

        if(waitpid(pid,&status,0)==-1)
          return -1;
        else
          return status;

      }

  }

}
```

Il programma completo è riportato qui di seguito:

```
#include "smallsh.h"

static char inpbuf[MAXBUF], tokbuf[2*MAXBUF],
*ptr = inpbuf, *tok = tokbuf; /* buffer e puntatori globali */
static char special[]={' ','\t','&',';','\n','\0'};

int userin(char *p)
{
  int c, count;
  ptr=inpbuf;
  tok=tokbuf;
  printf("%s",p); /* stampa il prompt */
  count=0;

  while(1)
    {

      if((c=getchar())==EOF)
        return(EOF);

      if(count<MAXBUF)
        inpbuf[count++]=c;

      if(c=='\n' && count<MAXBUF)
        {
          inpbuf[count]='\0';
          return count;
```

```
        }

      if(c=='\n') /* se linea troppo lunga, ricomincia */
        {
          printf("smallsh: input line too long\n");
          count=0;
          printf("%s",p);
        }

    }

}

int inarg(char c)
{
  char *p=special;

  while(*p!='\0')
    {

      if(c==*p++)
        return 0;

    }

  return 1;
}

int gettok(char **outptr)
{
  int type;
  *outptr=tok;

  while(*ptr==' ' || *ptr=='\t')
    ptr++;

  switch(*ptr) {
  case '\n':
    type=EOL;
    ptr++;
    break;
  case '&':
    type=AMPERSAND;
    ptr++;
    break;
  case ';':
    type=SEMICOLON;
    ptr++;
    break;
  default:
```

```
    type=ARG;

    while(inarg(*ptr))
      *tok++=*ptr++;

  }

  *tok++='\0';
  return type;
}

int runcommand(char **cline, int where)
{
  pid_t pid;
  int status;

  switch(pid=fork()) {
  case -1:
    perror("Errore nella chiamata alla fork");
    return -1;
  case 0:
    execvp(*cline,cline);
    perror(*cline);
    exit(1);
  default:

    if(where==BACKGROUND)
      {
        printf("Process id: %d\n",pid);
        return 0;
      }
    else
      {

        if(waitpid(pid,&status,0)==-1)
          return -1;
        else
          return status;

      }

  }

}

int procline(void)
{
  char *arg[MAXARG+1]; /* array di puntatori per runcommand */
  int toktype; /* typo del token */
  int narg; /* numero di argomenti correnti */
```

```
  int type; /* FOREGROUND o BACKGROUND */
  narg = 0;

  for(;;) /* ciclo infinito */
    {
      switch(toktype = gettok(&arg[narg])){
      case ARG: if(narg < MAXARG)
        narg++;
        break;
      case EOL:
      case SEMICOLON:
      case AMPERSAND:

        if(toktype == AMPERSAND)
          type = BACKGROUND;
        else
          type = FOREGROUND;

        if(narg != 0)
          {
            arg[narg] = NULL;
            runcommand(arg, type);
          }

        if(toktype == EOL)
          return;

        narg = 0;
        break;
      }

    }

}

char *prompt = "Command> "; /* prompt */

main()
{
  while(userin(prompt) != EOF)
    procline();
}
```