**1. 전체 코드의 흐름**

\* 코드 줄 번호를 나타내기 위해 [ ] 기호를 사용하였습니다.  
\* 코드에서 생략한 부분은 //omitted, 슈도코드 부분은 //pseudocode로 표현하였습니다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**main**함수입니다. **userin** 함수를 사용해 사용자에게 입력을 받고[6], 입력을 처리하는 **procline** 함수를 호출하는 것[7]을 반복합니다. 입력이 EOF일 때 종료합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**userin** 함수입니다. 인자로 넘어온 p 문자열을 출력하고[15] 사용자 입력을 inpbuf에 저장하는 역할을 합니다. getchar을 이용해 문자를 하나씩 입력 받으며[19] EOF일 때 EOF를 리턴하고[20] \n이 나올 때 멈춥니다. 입력 문자의 개수를 센 후[22] 그 수치가 입력 버퍼의 크기보다 크다면 경고 문구를 출력하고 재입력을 받게 합니다. [27~30]

텍스트이(가) 표시된 사진

자동 생성된 설명

**gettok** 함수입니다. 정의된 special character을 기준으로 token을 분리하여 인자인 outptr에 token단위로 저장하는 역할을 합니다. 먼저 입력 버퍼(ptr)의 앞 공백을 제거한 후[38~39] \n, &, ; 이 나타났을 때 type에다 기록합니다. [44~45] 그리고 inarg 함수를 호출하여 나머지 special character이 나타났을 때 type에다 ARG을 기록합니다. [49] special character의 위치에 \0을 넣음으로써 각각의 문자열 token으로 분리되게 합니다. [52]

텍스트, 장치, 조종판이(가) 표시된 사진

자동 생성된 설명

**inarg** 함수입니다. 인자로 넘겨받은 문자가 special character인지를 판별하는 역할을 합니다. 먼저 프로그램에 사용될 special character 배열을 정의하고[56], 그 배열을 탐색하면서 인자가 special character이면 0, 아니면 1을 리턴합니다. [59~63]

텍스트이(가) 표시된 사진

자동 생성된 설명

**procline** 함수입니다. token으로 분리된 명령어를 받아와 명령어가 요구하는 기능에 맞게 **runcommand**를 호출하는 역할을 합니다. 먼저 **gettok**을 호출해 arg 배열에 token을 받아옵니다.[69] 만약 token이 ARG라면 **gettok**를 한번 더 호출하게 합니다.[71~73] 그리고 token이 EOL, SEMICOLON, AMPERSAND 중 하나라면 token의 끝을 알리기 위해 NULL을 넣은 후[79] **runcommand**를 호출합니다.[81] token이 AMPERSAND이면 **runcommand**에 BACKGROUND를 넘기고[75], EOL이라면 **procline**을 종료합니다.[84]

**코드 수정)** runcommand 실행 도중 오류가 났을 경우를 처리하기 위해 조건문을 사용해 exit하는 code를 추가하였습니다. [81~82]

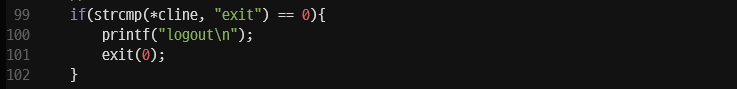
**2. 기능 구현**

텍스트이(가) 표시된 사진

자동 생성된 설명

**runcommand** 함수 내부에 모든 기능을 구현하였습니다. **runcommand** 함수는 명령을 토큰화한 배열(cline)을 인자로 받고[164] BACKGROUND/FOREGROUND 옵션을 받아 명령을 실행하는 역할을 합니다. 사용자 명령을 execvp 함수를 이용해서 실행한 후[175] 부모 프로세스에서 BACKGROUND이면 자식 pid 출력 후 리턴[181~184], FOREGROUND이면 자식 프로세스를 wait합니다.[185]

**2-1. exit 기능 구현**



cline 변수가 명령 토큰 배열이기 때문에 \*cline은 배열의 첫 번째 원소를 의미하게 됩니다. 이에 문자열을 비교하는 strcmp함수를 사용하여[99] 명령이 “exit”인지를 판단합니다. 만약 명령이 “exit”이라면 “logout”을 출력하고 exit(0)을 이용해 프로세스를 종료합니다. [101]

**2-2. cd 기능 구현**

텍스트이(가) 표시된 사진

자동 생성된 설명

2-1과 같은 원리로 \*cline을 이용해 명령이 “cd”인지를 판단합니다.[105] cd <dir>처럼 하나의 인자만 입력될 때에는 명령 토큰 배열(cline)의 끝, 즉 cline[2]가 NULL이어야 하므로 cline[2]가 NULL이 아닐 경우에는 관련 메시지를 출력하고 **runcommand**를 종료하게 합니다. 정상적인 입력일 때에는 chdir 함수를 사용해[110] current working directory를 바꾸고 **runcommand**를 종료합니다.

**2-3. > 기능 구현**

텍스트이(가) 표시된 사진

자동 생성된 설명

cline 변수가 명령 토큰 배열이기 때문에 ‘>’가 입력되면 배열 원소 중 하나로 저장되어 runcommand로 넘어옵니다. 따라서 strcmp를 사용한 반복문을 통해 토큰 배열에서 ‘>’의 위치를 찾아냅니다.[116-117] 그 후 ‘>’의 다음 인자, 즉 output redirection할 파일명을 open함수에 넘겨주어 파일을 엽니다(fd\_new).[118] 다음으로 ‘>’이 들어왔음을 알리는 변수 redirect를 1로 초기화하고[122] ‘>’의 값을 NULL로 초기화합니다.[123]

fork가 된 후에는 pid가 0일 때, 즉 자식 프로세스일 때 cline 배열을 vector 형태로 넘겨주게 됩니다.[137] ‘>’이 있음을 의미하는 redirect가 1일 때는 dup2를 사용해서 stdout의 file descriptor을 덮어쓰고 exec이 진행됩니다.[134] **이 때, ‘>’가 있는 곳이 NULL을 의미하게 되었으므로 exec 시에 ‘>’ 이전까지의 명령어를 수행하게 됩니다.** fork가 일어날 때 자식 프로세스는 부모 프로세스의 file descriptor도 복사하므로 **output redirection이 유지됩니다.**

**3. & 구현의 문제점 분석**

텍스트이(가) 표시된 사진

자동 생성된 설명

**문제점)** &가 입력되어 runcommand 함수의 where 인자에 BACKGROUND가 넘어가면, 부모 프로세스는 자식 프로세스의 pid를 출력하고 runcommand를 종료하게 됩니다. [98~101] 이 때, 그 아래의 if문[102]이 실행되지 않는데, 이것은 자식 프로세스를 wait하지 못하여 자식 프로세스가 zombie가 되는 결과를 낳게 됩니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 사진처럼 ls &를 입력하고 ps를 실행해 보면 자식 process인 ls가 background에서실행되다 zombie가 되어 defunct로 표시되는 것을 볼 수 있습니다.

**해결 방안)**

1. 부모 프로세스가 주기적으로 wait을 수행하는 부분이 필요할 것 같습니다.

2. init process는 주기적으로 wait을 수행하므로 runcommand를 종료하기 전에 자식 프로세스의 부모를 init으로 연결시켜 주는 과정이 필요할 것 같습니다.

**3. Discussion**

1. 왜 cd와 exit는 exec으로 수행되지 않는가?

cd와 exit은 쉘 내장 명령어입니다. 만약 cd가 프로세스라면 cd 프로세스 자신의 working directory를 바꾸고 종료되므로 사용하는 의미가 없습니다. 따라서 cd는 쉘 내부에서 존재해야 쉘의 current working directory를 바꾸어 가며 작업할 수 있습니다. exit도 마찬가지로 실행 중인 쉘 코드를 종료해야 하므로 쉘 내부에 존재해야 합니다.

2. defunct로 표시된 process에는 kill 명령어가 수행되지 않는가?

좀비 프로세스는 이미 끝난 프로세스여서 직접 kill되지 않고 wait으로만 거두어들일 수 있습니다.