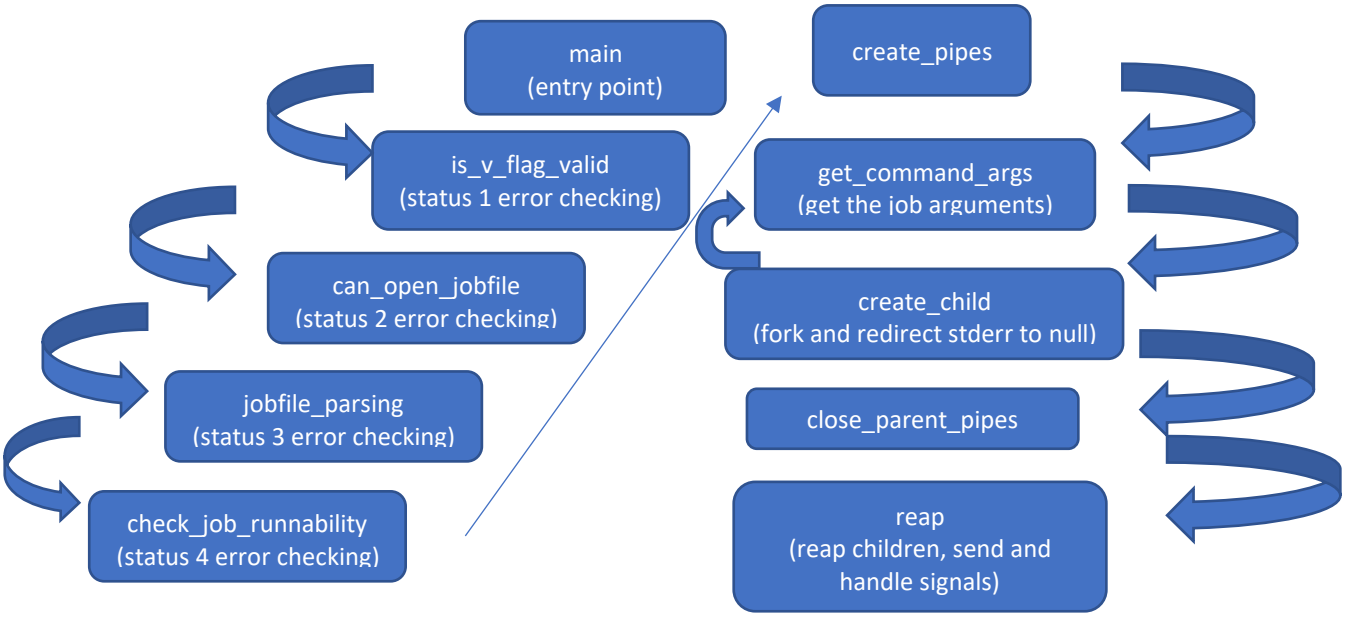
Design Document

Wenjun Yao, s4577546

- Overall program design
  In general, the program just follows the A3 specification sheet program operation flow. That is to say, processing the status 1 error first, then status 2, status 3, status 4. After all error checking is done, the parent process will create all the pipes before any forking, and after forking, all child processes will close unnecessary pipes and exec(). After all forkings are done, the parent process will close all open pipe ends on itself, and then start reaping, during which SIGABRT and SIGKILL will be sent to corresponding child process by the job's nature. During reaping stage, the parent also handles SIGHUP signal.

- Program flow



  The program will start at main, and then follow the arrows direction. After running check_job_runnability, the program will go to create pipes and then follow the arrows. Notice that get_command_args and create_child are ran in an iterative way. That is to say, after each time collecting a job line arguments, the create child will be ran until there is no more jobs.

- Use of data structures
  I use structs well throughout my assignment, for example my Argument struct contains the running mode (verbose mode or not), all the job files and its number. This allowed me to pass only one variable around (an instance of my argument struct) to functions, rather than 3 separate variables. This allows my code to be readable. Additionally, because I use

a argument struct, it allow me to organise my information to ensure that all relevant information is being passed.

- Use of enums
  I use the enum Mode in my code to store the running mode(verbose or not). This enum is also a member of the struct Argument. Enum is easy to compare values, saving me a lot of time using strcmp() to decide if the program is running in verbose mode if using a string to store running mode information instead. This also makes my code more readable.

- General style guide
  My code format follows CSSE2310/CSSE7231 C programming style guide. For example, all the function names are lowercase and use underscores to separate multiple words, and all type names begin with capital letters.

- Functional decomposition
  Functional decomposition is practiced in many places in my code. For example, in the reap() function, after a child is reaped, it should be removed from the parent's running child list. It is done by calling the remove_finished_process() function. Functional decomposition helps manage the length of a function and makes code easier to follow.

- Repeated code
  Yes, there is repeated code. At line 66 of runJobs.c, this block of code is actually repeated, but it has its unique effect and thus cannot be simply deleted. It can be fixed by replacing the line 61's condition by ***if (j == 1 | | j== 2)***.

- File handling
  File handling is also practiced at many places. For example, in the can_open_jobfile(), all file pointers that are not null after being assigned value by fopen() are closed. It is important remember to close open file pointers after file processing is done.