Homework #4

Question 1 (6 pt.) Shell

Extend data type `struct Command` with the following fields:

```
char *stdin_redirect;
char *stdout_redirect;
int background;
```

Fields `stdin_redirect` and `stdout_redirect` are strings indicating file names to which the standard input of the first sub-command and the standard output of the last sub-command will be redirected, respectively. Remember operators `<` and `>` are used in the shell to redirect the standard input and output, respectively. If any of these redirections is not provided by the user, the corresponding field in the structure will take a value of `NULL`.

Field `background` is a flag indicating whether the process should run in the background. If the user adds the `&` operator at the end of the line, `background` should be set to 1. Implement the following functions, or modify the previous versions of them from previous assignments:

a) `void ReadRedirectsAndBackground(struct Command *command);`

   This function populates fields `stdin_redirect`, `stdout_redirect`, and `background` for the `command` passed by reference in the first and only argument. The function assumes that all other fields of the `command` structure have already been populated, as a result to a previous invocation to `ReadCommand`. The function should internally scan the arguments from the last sub-command in reverse order, extracting trailing `&`, `> file`, or `< file` patterns in a loop.

b) `void PrintCommand(struct Command *command);`

   Extend this function to dump the value of `stdin_redirect`, `stdout_redirect`, and `background` for the command passed by reference in its first and only argument. Make sure that you consider the special cases where the redirection fields are set to `NULL`.

c) Write a main program that reads a command line from the standard input, creates the command object, and dumps all its fields, by invoking the previous functions.

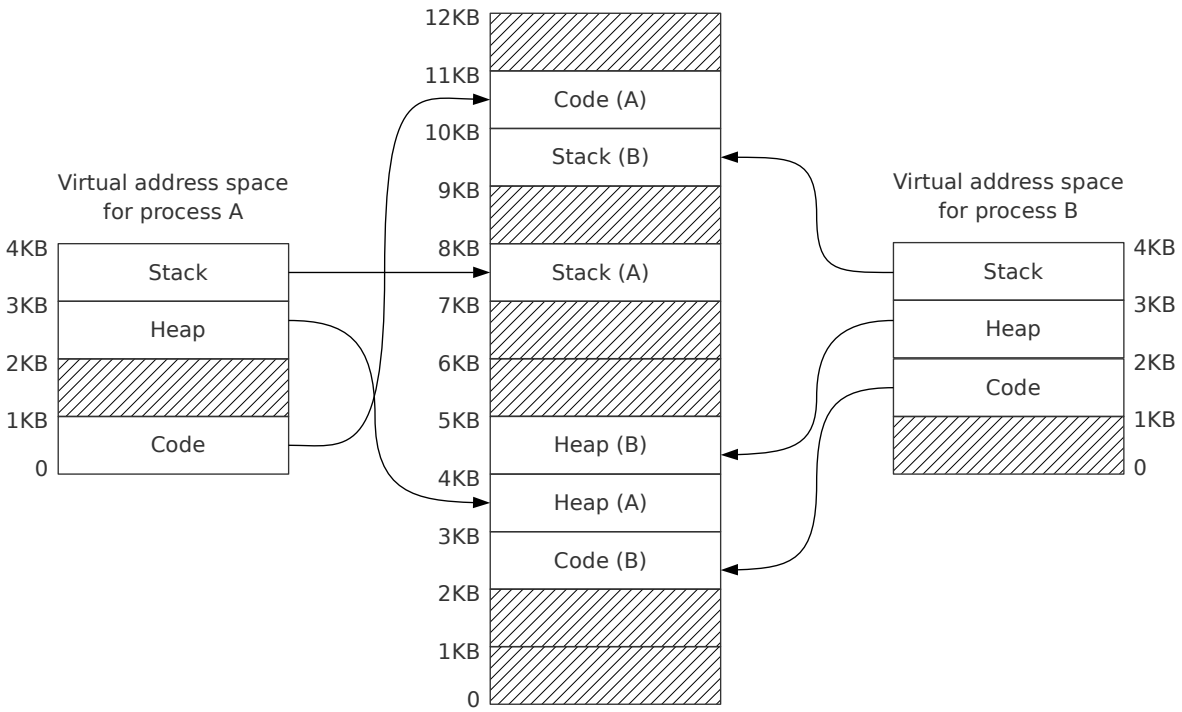
This is an example of the execution of your program:

```
$ ./main
Enter command: a | b c > output.txt < input.txt &

Command 0:
argv[0] = 'a'

Command 1:
argv[0] = 'b'
argv[1] = 'c'

Redirect stdin: input.txt
Redirect stdout: output.txt
Background: yes
```

Write your program in a file called `main.c` and place it in a directory named `q1`. Create a ZIP package named `q1.zip` containing directory `q1` and upload it on Canvas.

Question 2 (4 pt.)

Consider a machine with a 12KB physical memory space running an OS that uses a 4KB virtual memory space for its processes. The machine is executing two processes (A and B), where each process has 3 allocated segments with the following mappings:



a) (1 pt.) Show the state of a basic segment table containing the information represented in the diagram above. The table should include the following columns:

- *Segment*. The segment index, starting at 0. Sort segments by process (first A, then B), and by base virtual address (lower virtual address first).
- *Process*. Process identifier.
- *Virtual address*. Base address in virtual address space.
- *Size*. The segment size.
- *Physical address*. Base address in physical address space.

b) (3 pt.) Perform a set of address translations based on the segment table above. Use a table with the following columns to represent the state of each memory translation:

- *Process*. Process accessing memory, as given below.

- *Virtual address*. Virtual address access by the process, as given below.
- *Segment*. Index of the segment affected by this access, or "–" for segmentation fault.
- *Offset*. Offset within the segment, or "–" for segmentation fault.
- *Physical address*. Final physical address, or "Segfault" for segmentation fault.

Use the following memory accesses:

- Process A, address 2148
- Process A, address 1324
- Process A, address 4095
- Process B, address 512
- Process B, address 2048
- Process B, address 3272