The number of process we created is 11 in our example.

The format of this description file should strictly follow like this:
"A 2 B C\nB 1 D\nC 0\nD 0\0" where the end of a line and the beginning of a new line is only separated by ‘\n’ and the end of file ‘\0’ is included.

Such a number of processes is acceptable for my laptop because it does not exceed the maximum number of processes my laptop can maintain.

The appearance of printing result will be like this:

```
[lc893@engsoft:~$ cd computer_system/problem1
[lc893@engsoft:~/computer_system/problem1$ gcc -std=c99 -o problem1 problem1.c
[lc893@engsoft:~/computer_system/problem1$ ./problem1
Data structure tree:
A1
        B1
                C1
                C2
                C3
        B2
                C4
        B3
        B4
                C5
                C6
Process tree:
A1
        B1
                C1
                C2
                C3
        B2
                C4
        B3
        B4
                C5
                C6
```

Each column represents the level of the tree.

The program will first create a tree data structure by calling create_tree() which will return a pointer to the root node of the tree. Then we use print_tree(root) to print this tree structure.

Second, pass the root node pointer to the create_process_tree(root) function to create the desired process tree and print all process name in the desired order in this function.

If the process has no children, it will terminate after it sends a message its parent using write() and then call exit() to terminate.

If the process has children and is not the root, it will wait until all its children have terminated using wait() and call exit().

If the process is the root, it will terminate until all its children have terminated using wait();

The process tree will be created and printed in a recursive way:

For example:

The creation order of processes is unexpected.
The printing order is like this: A1->B1->C1->C2->C3->B2->C4->B3->B4->C5->C6

The tree is given like this: (the first column contains all tree nodes and the second column represents the number of children the certain node can have. All children names of a certain node are listed after second column)

```
A1 4 B1 B2 B3 B4
B1 3 C1 C2 C3
B2 1 C4
B3 0
B4 2 C5 C6
C1 0
C2 0
C3 0
C4 0
C5 0
C6 0
```

The recursion is like this:

```
void create_process_tree(struct tree_node *root, int level){
    /*used for parent to write info to child and child to read info
from parent
    but child is not supposed to have right to write to parent and
vice versa
    */
    int fd1[2];

    /*used for child to write info to parent and parent to read info
from child
    but parent is not supposed to have right to write to child and
vice versa
    */
    int fd2[2];
    int n = root->children_num;
    int writefd[n];
    int readfd[n];
    pid_t pid[n];
    if (n==0){
```

```
        for (int i = 0; i < level; i++){
            printf("\t");
        }
        printf("%s\n", root->name);
        return;
    }
    for (int i =0; i<n; i++){
        pipe(fd1);
        pipe(fd2);
        pid[i] = fork();
        if (pid[i] == 0){
           /*child is not supposed to write to parent using pipe1*/
            close(fd1[1]);
            /*read message from parent, the message content doesn't
matter*/
            read(fd1[0],&pid[i],sizeof(pid_t));
            create_process_tree(root->children[i], level+1);//
recursively calling create_process_tree
            /*child is not supposed to read from parent using pipe2*/
            close(fd2[0]);
            /*send message to parent, the message content doesn't
matter*/
            write(fd2[1],&pid[i],sizeof(pid_t));
            exit(0);
        }
        /*parent is not supposed to read from child using pipe1*/
        close(fd1[0]);
        writefd[i] = fd1[1];
        /*parent is not supposed to write to child using pipe2*/
        close(fd2[1]);
        readfd[i] = fd2[0];
    }
```