```c
/*--------------------------------------------------------
UNIX Shell Project
function prototypes, macros and type declarations for job_control module

Sistemas Operativos
Grados I. Informatica, Computadores & Software
Dept. Arquitectura de Computadores - UMA

Some code adapted from "Fundamentos de Sistemas Operativos", Silberschatz et al.
--------------------------------------------------------*/

#ifndef _JOB_CONTROL_H
#define _JOB_CONTROL_H

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <termios.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

// ----------- ENUMERATIONS ------------------------------------------
enum status { SUSPENDED, SIGNALED, EXITED, CONTINUED};
enum job_state { FOREGROUND, BACKGROUND, STOPPED, RESPAWNABLE };
static char* status_strings[] = { "Suspended", "Signaled", "Exited", "Continued"};
static char* state_strings[] = { "Foreground", "Background", "Stopped", "Respawnable" };

// ----------- JOB TYPE FOR JOB LIST ---------------------------------
typedef struct job_
{
 pid_t pgid; /* group id = process lider id */
 char * command; /* program name */
 char** args;
 enum job_state state;
 struct job_ *next; /* next job in the list */
} job;

// ----------- COMMAND TYPE FOR HISTORY LIST -------------------------------


typedef struct history_{
    const char * command;
    char** args;
    enum job_state state;
 struct history_ *next;
 struct history_ *prev;
 struct history_ *last;
} history;


// ----------- TYPE FOR JOB LIST ITERATOR ----------------------------
typedef job * job_iterator;

// ------------------------------------------------------------------
//      PUBLIC FUNCTIONS
// ------------------------------------------------------------------

void get_command(char inputBuffer[], int size, char *args[],int *background, int *respawnable);

job * new_job(pid_t pid, const char * command, enum job_state state);

void add_job(job * list, job * item);
void add_command(history ** hist, char **args, enum job_state estado);

void add_job_respawnable(job* list, job* item, char** args);
```

```
int delete_job(job * list, job * item);

job * get_item_bypid(job * list, pid_t pid);

job * get_item_bypos(job * list, int n);

enum status analyze_status(int status, int *info);


// History

history * new_history();

void print_history(history * his);

history * get_com_bypos(history * his, int n);

// ---------------------------------------------------------------------
//      PRIVATE FUNCTIONS: BETTER USED THROUGH MACROS BELOW
// ---------------------------------------------------------------------

void print_item(job * item);


void print_list(job * list, void (*print)(job *));

void terminal_signals(void (*func) (int));

void block_signal(int signal, int block);

// ---------------------------------------------------------------------
//      PUBLIC MACROS
// ---------------------------------------------------------------------

#define list_size(list)    list->pgid   // number of jobs in the list
#define empty_list(list)   !(list->pgid)  // returns 1 (true) if the list is empty

#define new_list(name)     new_job(0,name,FOREGROUND)  // name must be const char *

#define get_iterator(list)   list->next // return pointer to first job
#define has_next(iterator)   iterator
#define next(iterator)       ({job_iterator old = iterator; iterator = iterator->next; old;})

#define print_job_list(list)   print_list(list, print_item)


#define restore_terminal_signals()   terminal_signals(SIG_DFL)
#define ignore_terminal_signals() terminal_signals(SIG_IGN)

#define set_terminal(pid)        tcsetpgrp (STDIN_FILENO,pid)
#define new_process_group(pid)   setpgid (pid, pid)

#define block_SIGCHLD()     block_signal(SIGCHLD, 1)
#define unblock_SIGCHLD()   block_signal(SIGCHLD, 0)

// macro for debugging----------------------------------------------------
// to debug integer i, use:    debug(i,%d);
// it will print out:  current line number, function name and file name, and also variable name, value and type
#define debug(x,fmt) fprintf(stderr,"\"%s\":%u:%s(): --> %s= " #fmt " (%s)\n", __FILE__, __LINE__, __FUNCTION__, #x, x, #fmt)

// ---------------------------------------------------------------------
#endif
```