```c
//
//  main.c
//  All
//
//  Created by Filippo Fontanelli on 02/04/11.
//  Copyright 2011 __MyCompanyName__. All rights reserved.
//

#include <stdio.h>
#include "main.h"



int main (int argc, char * argv[])
{
    int end;/*variabile utilizzata per la terminazione del comando*/
    char *app;/*char* utilizzato per memorizzare i vari token di caratteri
        letti da stdin*/
    int endstdin;/*utilizzata per la terminazione della lettura da stdin*/
    char* pch;/*variabile utilizzata per l'output della funzione strtok*/
    queue *q;
    token *t;
    int n , endline;
    int want_opt [4];
    char delimit[] = "[:]";/*array utilizzato come parametro della funzione
        strok*/

    OptMode mode;
    ReadMode rmod;

    /*Parsing linea di comando*/
    if(check_option_cmd_line(argc,argv,want_opt))
        return EXIT_FAILURE;

    /*Verifico le opzioni impostate nel comando*/
    if(want_opt[O_ASCI])
        mode = O_ASCI;
    else if(want_opt[O_ANSI])
        mode = O_ANSI;
    else
        mode = O_DEFAULT;

    if(want_opt[O_7]){
        rmod = R_7;
    }else
        rmod = R_DEFAULT;

    /*Dopo aver effettuato il parsing ed impostato mode procedo con la
        scansione del testo ricevuto da stdin*/
    end = 0;
    endstdin = 0;
    endline = 0;
    MALLOC(app, sizeof(char)*want_opt[O_COLUM]*2);//raddoppio la dimensione per
        evitare segmentation fault

    while(!end){
        memset(app, '\0', strlen(app));

        end = Read[rmod](app,want_opt[O_COLUM]);

        MALLOC(q, sizeof(struct queue))
        INIT_QUEUE(q,want_opt)

        /*Utilizzando la funzione srtok procedo alla scanzione dell'input*/
```



```c
        pch = strtok(app,delimit);

        while(pch != NULL){

            MALLOC(t, sizeof(struct token));
            INIT_TOKEN(t);

            if(CheckFunc[mode](t,q,pch) == 0)
                strcat(t->stringa, pch);

            /*conto il numero di spazi presenti nel testo*/
            for(n = 0; pch[n] != '\0'; n++)
                t->n_space += (pch[n] == ' ');

            /*inserisco il token all'interno della coda*/
            enqueue(t, q);

            pch = strtok (NULL, delimit);
        }

        /*giustifico e stampo i token della coda*/
        giustifica(q);
        PrintFun[mode](q,&endline);

        free_queue(q);
        free(pch);
    }

    free(app);

    return 0;
}

/*Funzione fittizia che non fa nulla*/
int CheckDefault(token *t, queue *q, char *pch){
    return 0;
}

int PrintDefault(queue *q, int *endline){
    token *t;
    int sum,n;
    t = front(q);
    sum = 0;

    while (t != NULL) {
        n = 0;
        while(t->stringa[n] != '\0'){
            if ( t->stringa[n] == '\n')
                *endline= 0;
            if (*endline == q->want_opt[O_COLUM]+1 ){
                fprintf(stdout,"%c", '\n');
                *endline = 0;
            }
            fprintf(stdout,"%c",t->stringa[n]);
            n++;
            (*endline)++;
        }
        sum += strlen(t->stringa);
        t = t->next;
    }
    return sum;
}

int check_option_cmd_line(int argc, char** argv,int *option){
```

```
    int i;
    char ch;

    if (argv == NULL) return ERROR;

    /*Inizializzazione dei parametri relativi alle opzioni del comando*/
    for (i = 0; i < OPT_NUM; i++) {
        option[i] = 0;
    }
    option[O_COLUM] = 78;

    while ((ch = getopt (argc,argv,"w:ax7"))!=EOF)
    {
        switch (ch)
        {
            case 'a':
                if (option[O_ANSI]) {
                    fprintf(stderr,"Errore, parametro -a incompatibile
                        contemporaneamente al parametro -x!!\n");
                    exit(EXIT_FAILURE);
                }
                option[O_ASCI] = 1;
                break;
            case 'x':
                if (option[O_ASCI]) {
                    fprintf(stderr,"Errore, parametro -x incompatibile
                        contemporaneamente al parametro -a!!\n");
                    exit(EXIT_FAILURE);
                }
                option[O_ANSI] = 1;
                break;
            case 'w':
                option[O_COLUM] = atoi(optarg);
                break;
            case '7':
                option[O_7] = 1;
                break;
            default:
                fprintf(stderr,"Uso: %s [-a] [-x] [-w] [-7] \n", argv [0]);

        }
    }

    return 0;

}

int giustifica(queue *q){
    token *t;
    int i,x,s,space_lost;
    char * app;
    if (q->length_string > q->want_opt[O_COLUM])
        return 0;
    else{

        if (strpbrk(rear(q)->stringa,"\n") != NULL)
            return 0;
        else{

            t= front(q);
            space_lost = q->want_opt[O_COLUM] - q->length_string;

            while(t->next != NULL){
                MALLOC(app, sizeof(char)*q->want_opt[O_COLUM]+1);
```



```
                for(i = 0,s =0;t->stringa[i] != '\0';i++){
                    if (t->stringa[i] == ' '){
                        if (x != 0){
                        app[s++] = ' ';
                        space_lost--;
                        if(space_lost == 0)
                            break;
                        }
                        x++;
                    }
                    app[s++] = t->stringa[i];
                }
                app[s] = '\0';
                memset(t->stringa, '\0', strlen(t->stringa));
                strcpy(t->stringa, app);
                free(app);
                t = t->next;
            }

        }

    }

    return 0;

}


int ReadStdinDefault(char * app, int colum){
    int n, endstdin;
    char c;
    n = 0;
    endstdin = 0;

    while(!endstdin && (c = fgetc(stdin)) != EOF){
        if(n >= colum-1 && c == ' ')
            endstdin = 1;
            app[n++] = c;
    }
    app[n]='\0';

    return (c == EOF);
}

int ReadStdin7(char * app, int colum){
   /* int n,i, endstdin;
    char c,tmp;
    char *x;
    wchar_t arr[6];


    setlocale(LC_ALL,"");


    n = 0;
    i = 0;
    endstdin = 0;

    MALLOC(x, sizeof(char)* colum + 1);
    while(!endstdin && (c = fgetc(stdin)) != EOF){
        if(n >= colum-1 && c == ' ')
            endstdin = 1;
        if (!isascii(c)){
            x[i] = '\0';
```

```
            strcat(app, x);
            x[0]= c;
            x[1] = '\0';
            tmp = mbtowc(arr, x , 6);
           strcat(app,(char *)arr);
            fprintf(stdout,"wide-character string: %s\n",(char *)arr);
        }else
            x[i++] = c;
        n++;
    }

    x[n]='\0';
    strcat(app, x);
    if(c == EOF)
        return 1;*/

    return ReadStdinDefault(app, colum);
}
```