```c
/*Completare il programma C nelle parti indicate con TODO (lasciando inalterate le altre
parti) in modo che:

 legga da standard input una stringa;
 stampi l'intera stringa;
 stampi la stringa dal settimo carattere (indice 6) fino alla fine, se la stringa più
corta di sette caratteri non stampa nulla.


*/


/**
 * Implementare una stringa come lista di elementi di tipo node_t terminata da STRING_END.
 * completare tutte le funzioni dichiarate ma non definite nel programma.
 */
#include <stdio.h>
#include <stdlib.h>  // malloc(), free(), exit()

#define STRING_START 0
#define STRING_END -1

struct node_t {
    char value;
    struct node_t *next;
};

void check_alloc(struct node_t *p);
struct node_t *make_string(char *s);
void print_string(struct node_t *head, int start, int end);
void free_string(struct node_t *head);

int main() {
    struct node_t *s;
    char *buffer = NULL;
    long unsigned int len = 80;
    getline(&buffer, &len, stdin);      //quando stampo considero anche lo \n

    s = make_string(buffer);

    print_string(s, STRING_START, STRING_END);      //STRING_START = 0      STRING_END =
-1

    print_string(s, 6, STRING_END);

    print_string(s, 2, 4);

    free_string(s);
    return 0;
}

/**
 * Controlla se un puntatore è valido e se non è così
 * stampa la stringa "Errore nell'allocazione della memoria.\n" ed esce con exit(-1).
 */
void check_alloc(struct node_t *p) {
    // TODO

    if( p == NULL ){

        printf("Errore nell'allocazione della memoria.\n");
        exit(-1);       //fa terminare il programma     -->     si per far terminare TUTTO
il programma
    }
```

```c
}

/**
 * Crea una lista concatenata di nodi di tipo node_t
 * a partire dalla stringa s.
 * Se la stringa s è vuota, ritorna NULL.
 */
struct node_t *make_string(char *s) {       // *s = (nel main) a buffer     (buffer è un puntatore / contenitore e contiene la strimnga inserita dall'utente a mano dal terminale)
    // TODO

    // se s è vuota     -->     controllo il 1 carattere
    if( s == NULL || (*s) == '\n' || *s == '\0' ){

        return NULL;
    }

    long node_size = sizeof(struct node_t);

    struct node_t *testa = ( struct node_t * ) malloc ( node_size );
    check_alloc(testa);
    testa -> value = *s;
    testa -> next = NULL;

    struct node_t *coda = testa;

    do{
        s++;        //incrementiamo il punttore -->     indirizzo a cui punta   --> caratetre successivo

        struct node_t *nuovo_blocco = ( struct node_t * ) malloc ( node_size );
        check_alloc(nuovo_blocco);
        nuovo_blocco->value = *s;
        nuovo_blocco->next = NULL;

        coda->next = nuovo_blocco;
        coda = nuovo_blocco;

    } while ( *s != '\n' );



    return testa;
}

/**
 * Stampa i caratteri di una lista concatenta di tipo node_t dall'indice start all'indice end inclusi.
 * Se end è STRING_END, stampa i carattteri fino alla fine della lista.     (STRING_END = -1)
 * Si ipotizza che start sia maggiore di zero e end sia maggiore di zero oppure sia STRING_END.
 */
void print_string(struct node_t *head, int start, int end) {
    // TODO

    int stampa = 0;

    int i = 0;      //conta a quale carattere siamo arrivati

    do{
        if( i >= start && ( end == STRING_END || i <= end ) ){

            printf("%c", head -> value);        //head = novo_blocco        ?? (in teoria)

        }
```

```
        i++;

        head = head->next;

    } while( head != NULL);

    printf("\n");

}

/**
 * Libera con free() la memoria della lista concatenata che inizia con head.
 */
void free_string(struct node_t *head) {
    // TODO

    struct node_t *salvataggio;

    while( head != NULL ){
        salvataggio = head -> next;

        free(head);
        head = salvataggio;

    }


}
```