```c
/*
 * Implementação das listas que vamos usar no projeto.
 * Estas são listas de payload genérico, doubly-linked com dummy head node.
 *
 * A desvantagem de utilizar listas de payload genérico é que
 * obrigamos o utilizador a fazer cast do payload para o que ele
 * precisar, o que pode ser chato, adicionar complexidade ao código
 * e/ou criar bugs difíceis de perceber se nos esquecermos de
 * fazer cast da payload... (ou seja, fazia-se dereference dum
 * void pointer, o que é ilegal) _E_ o utilizador tem de fazer free()
 * da payload manualmente.
 *
 * Mas, por outro lado, o programa torna-se mais modular,
 * mais fácil de compreender e abstrai-se assim toda a parte
 * das listas.
 *
 */
#include <stdlib.h>
#include <stdio.h>
#include "list.h"
#include "error.h"

// Aceder a um nó numa posilão especifica da lista
List *list_follow(List *head, int pos)
{
    if (pos < 0) {
        fprintf(stderr, "Erro: tentou-se aceder a um nó não existente na lista.\n");
        exit(EXIT_FAILURE);
    }

    List *aux = head;
    for (int i = 0; i < pos; i++) {
        if (aux != NULL) {
            aux = aux->next;
        }
        else {
            fprintf(stderr, "Erro: tentou-se aceder a um nó não existente na lista.\n");
            exit(EXIT_FAILURE);
        }
    }

    return aux;
}

// Inserir um nó numa posição especifica da lista
void list_insert_pos(List *head, int pos, void *payload)
{
    List *aux = list_follow(head, pos - 1);

    List *new = (List *) ecalloc((size_t) 1, sizeof(List));
    new->payload = payload;

    new->next = aux->next;
    if (aux->next != NULL)
        aux->next->prev = new;
    else {
        // inserting at the tail, no need to set aux->next->prev
    }

    new->prev = aux;
    aux->next = new;
}


// Inserir na tail
void list_append(List *head, void *payload)
{
    List *aux = head;
```

```c
    while (aux->next != NULL)
        aux = aux->next;
    List *tail = aux;

    List *new_tail = (List *) ecalloc((size_t) 1, sizeof(List));

    new_tail->payload = payload;

    new_tail->next = NULL;
    new_tail->prev = tail;
    tail->next = new_tail;
}

// Remover um nó da lista
void *list_remove(List *node)
{
    List *to_rm = node;
    void *payload = to_rm->payload;

    if (node->next != NULL) {
        node->next->prev = to_rm->prev;
    }
    else {
        // removing tail, it has no next, skip.
    }

    if (node->prev != NULL) {
        node->prev->next = to_rm->next;
    }
    else {
        fprintf(stderr, "Erro: tentou-se remover o dummy head node da lista.\n");
        exit(EXIT_FAILURE);
    }

    free(to_rm);
    return payload;
}


// Remover um nó especifico da lista
void *list_remove_pos(List *head, int pos)
{
    List *to_rm = list_follow(head, pos);
    void *payload = list_remove(to_rm);
    return payload;
}
```