- Inserimento in Coda;
- Funzione Lunghezza;
- Funzione trova;
- Funzione che conta le occorrenze di un valore intero x in una lista di interi
- Funzione che elimina l'ultima occorrenza di un elemento da una lista
- Date due liste L1 ed L2 verifica se tutti gli elementi di L1 sono in L2;
- Date due liste L1 ed L2 verifica se L1 è una sottolista di L2

```
typedef struct lista {
    int value;
    struct lista* next;
} lista;

typedef lista* listaPtr;
```

- Inserimento in Coda;

```
listaPtr insertTail(int v, listaPtr h, listaPtr* tail) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;
    if (h == NULL) {
        h = x;
        *tail = x;
    }
    else {
        (*tail)->next = x;
        *tail = x;
    }
    return h;
}

void insertTail(int v, listaPtr* h, listaPtr* tail) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;
    if (*h == NULL) {
        *h = x;
        *tail = x;
    }
    else {
        (*tail)->next = x;
        *tail = x;
    }
}

listaPtr insertTail(int v, listaPtr h) {
    listaPtr x = (listaPtr) malloc(sizeof(lista));
    x->value = v;
    x->next = NULL;
    if (h == NULL) {
        h = x;
    }
    else {
        listaPtr it = h;
        listaPtr prev = it;
        it = it->next;
        while (it != NULL) {
            prev = it;
            it = it->next;
        }
```

```
        prev->next = x;
    }
    return h;
}
```

- Funzione Lunghezza;

```
int listLength (listaPtr h) {
    int count = 0;
    while (h != NULL) {
        count++;
        h = h->next;
    }
    return count;
}
```

- Funzione trova;

```
listaPtr findValue (listaPtr h, int x) {
    listaPtr aux = NULL
    while (h != NULL) {
        if (h->value == v) {
            return h;
        }
    }
    return aux;
}
```

- Funzione che conta le occorrenze di un valore intero x in una lista di interi

```
int contaOccorrenze (listaPtr h, int v) {
    int count = 0;
    while (h != NULL) {
        if (h->value == v) {
            count++;
        }
        h = h->next;
    }
    return count;
}
```

- Funzione che elimina tutte e l'ultima occorrenza di un elemento da una lista

```
listaPtr eraseAll(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    listaPtr aux;
    listaPtr prev;
    prev = h;
    listaPtr it = h->next;
    while (it != NULL) {
        printf("analizzo elemento %d\n", it->value);
        if (it->value == v) {
            printf("entro\n");
            listaPtr aux = it;
            prev->next = (it->next);
            free(aux);
            it = prev->next;
        }
        else {
```

```
            prev = it;
            it = it->next;
        }
    }
    if (h->value == v) {
        printf("entro2\n");
        return h->next;
    }
    printf("esco\n");
    return h;
}

void eraseAllRic(listaPtr* h, int v) {
    listaPtr temp;
    if (*h == NULL) {
        return;
    }
    if ((*h)->value == v) {
        temp = *h;
        *h = (*h)->next;
        free(temp);
        eraseAllRic(h, v);
        return;
    }
    eraseAllRic(&((*h)->next), v);
    return;
}

listaPtr eraseAllRic2(listaPtr h, int v) {
    listaPtr temp;
    if (h == NULL) {
        return h;
    }
    if ((h)->value == v) {
        temp = h;
        h = (h)->next;
        free(temp);
        return eraseAllRic2(h, v);
    }
    h->next = eraseAllRic2(h->next, v);
    return h;
}

listaPtr eraseLast(listaPtr h, int v) {
    if (h == NULL) {
        return h;
    }
    int index  = -1;
    int count = 1;
    listaPtr it, aux, prev;
    prev = h;
    it = h->next;
    listaPtr check = NULL;
    while (it != NULL) {
        printf("analizzo elemento %d\n", it->value);
        if (it->value == v) {
            check = prev;
        }
        prev = it;
        it = it->next;
    }
```

```
    if (h->value == v && check == NULL) {
        aux = h;
        h = h->next;
        free(aux);
        return h;
    }
    if (check != NULL) {
        aux = check->next;
        check->next = (check->next)->next;
        free(aux);
    }
    return h;
}
```

- Date due liste L1 ed L2 verifica se tutti gli elementi di L1 sono in L2;

```
int checkElements(listaPtr L1, lista Ptr L2) {
    if (L1 == NULL) {
        return 1;
    }
    while (L1 != NULL) {
        if (findValue(L2, L1->value) == 0) {
            return 0;
        }
        L1 = L1->next;
    }
    return 1;
}
```

- Date due liste L1 ed L2 verifica se L1 è una sottolista di L2 (L1 compare all'interno di L2

```
int listPrefix (listaPtr L1, lista Ptr L2) {
    if (L1 == NULL) {
        return 1;
    }
    while (L1 != NULL && L2 != NULL) {
        if (L1->value != L2-> value) {
            return 0;
        }
        L1 = L1->next;
        L2 = L2->next;
    }
    if (L2 == NULL) {
        if (L1 == NULL) {
            return 1
        }
        return 0;
    }
    return 1;
}

int checkSubList(listaPtr L1, lista Ptr L2) {
    if (listLength(L1) > listLength(L2)) {
        return 0;
    }
    if (L1 == NULL) {
        return 1;
    }
    while (L2 != NULL) {
        if (listPrefix(L1, L2)) {
```

```
            return 1;
        }
        L2 = L2->next;
    }
    return 0;
}
```