```c
int hashInsert(HashTable T, int k){
    for(int i=0; i<T->dim; i++){
        int pos = linearProbing(k, i, T->dim);
        if(pos >= T->dim) return -2;
        // se è gia presente ritorna -1
        if(T->array[pos] == k) return -1;
        // inserimento
        if(T->array[pos] == -1){
            T->array[pos] = k;
            return pos;
        }
    }
    return -2;
}
```

```c
int hashSearch(HashTable T, int k){
    for(int i=0; i<T->dim; i++){
        int pos = linearProbing(k, i, T->dim);
        if(pos >= T->dim) return -2;
        // se è presente ritorna l'elemento
        if(T->array[pos] == k) return T->array[pos];
        // se non è presente ritorna -1
        if(T->array[pos] == -1) return -1;
    }
    return -2;
}
```

```
list ordInsert(int k, list p, list l) {
    if (l == NULL) { // la lista è vuota
        return Cons(k, p, NULL);
    } else if (k == l->info) {
        // Element already exists, don't insert duplicate
        return l;
    } else if (k < l->info) { // k < l->info
        return Cons(k, p, l);
    } else { // k > l->info
        l->next = ordInsert(k, l, l->next); // Insert aft
        return l;
    }
}
void hashInsert(HashTable T, int k) {
    int pos = hashFun(k, T->dim);
    T->array[pos] = ordInsert(k, NULL, T->array[pos]);
}
```

```
int hashSearch(HashTable T, int k) {
    int pos = hashFun(k, T->dim);
    while(T->array[pos] != NULL){
        if (T->array[pos]->info == k) return T->array[pos]->info;
        else T->array[pos] = T->array[pos]->next;
    }
    return -1;
}
```