```
template <class Item>
class node
{
public:
    // TYPEDEF
    typedef Item value_type;
    // CONSTRUCTOR
    node(const Item& init_data=Item( ), node* init_link=NULL)
        { data_field = init_data; link_field = init_link; }
    // MODIFICATION MEMBER FUNCTIONS
    Item& data( ) { return data_field; }
    node* link( ) { return link_field; }
    void set_data(const Item& new_data) { data_field = new_data; }
    void set_link(node* new_link) { link_field = new_link; }
    // CONST MEMBER FUNCTIONS
    const Item& data( ) const { return data_field; }
    const node* link( ) const { return link_field; }
private:
    Item data_field;
    node *link_field;
};

// FUNCTIONS to manipulate a linked list:
template <class Item>
void list_clear(node<Item>*& head_ptr);

template <class Item>
void list_copy
    (const node<Item>* source_ptr, node<Item>*& head_ptr, node<Item>*& tail_ptr)

template <class Item>
void list_head_insert(node<Item>*& head_ptr, const Item& entry);

template <class Item>
void list_head_remove(node<Item>*& head_ptr);

template <class Item>
void list_insert(node<Item>* previous_ptr, const Item& entry);

template <class Item>
    std::size_t list_length(const node<Item>* head_ptr);

template <class NodePtr, class SizeType>
NodePtr list_locate(NodePtr head_ptr, SizeType position);

template <class Item>
void list_remove(node<Item>* previous_ptr);

template <class NodePtr, class Item>
NodePtr list_search(NodePtr head_ptr, const Item& target);
```