Consider the following structure declaration for a general directed graph data structure. In this structure, the size field gives the number of outgoing edges, and the children field is a pointer to an array of pointers to the child nodes.

```c
typedef struct node Node;
struct node {
    bool flag;
    int size;
    Node **children; // pointer to an array of Node pointers
};
```

(a) Define a function `Node *node(int n, Node **children)` which builds a new node from its arguments, taking ownership of the children argument and initializing the flag field to false. [2 marks]

(b) Write a function `Node *example(void)` which returns a new graph with the following structure, with the return value corresponding to n1: [2 marks]

```
\[ \text{n}_1 \rightarrow \text{n}_2 \]
```

(c) Define a structure for representing a linked list of `Node *` pointers, with a `Nodelist` typedef for the structure. [2 marks]

(d) Supposing we represent the empty linked list with the `NULL` pointer, and a cons cell with a pointer to a `Nodelist`, define a function `Nodelist *cons(Node *head, Nodelist *tail)` to add an element to this linked list. [2 marks]

(e) Write a function `Nodelist *reachable(Node *node)` which returns a list of all the nodes reachable from the argument node, including node itself. This list should contain every reachable node, and have no duplicates. You may assume that the flag field of every reachable node is set to `false` on entry to this function, and that your routine may modify it as you wish. [7 marks]

(f) Define a function `void free_node(Node *node)` which deallocates all the node objects reachable from the argument node. You may assume that the flag field of every reachable node is set to `false` on entry to this function, and that your routine may modify it as you wish. [5 marks]