Introduction to a singly-linked list

A singly-linked list is the most basic of all linked data structures. It is simply a sequence of dynamically allocated nodes, each of which refers to its successor node in the list. (The successor field of the last node is NULL.) Following is the conceptual diagram of a non-empty list.

The list is accessible by means of the pointer to the first node, being referred to here as head. A node itself consists of two fields, as shown below.

For simplicity, the info field in the present discussion is assumed to be of type int. Accordingly, the following declaration captures the blueprint of a node:

```c
struct nodeType
{
    int info;
    nodeType* link;
};
```

The accompanying implementation consists of functions buildList, printList, length and isPresent along with the test program main. Here is a brief description of the four functions.

```c
nodeType* buildList();
// This function reads a sequence of integers ending with -999, builds
// a linked list of nodes holding those integers (excluding -999) in
// order, and returns the pointer to the first node of the list.
```
In order to efficiently build a list, this function (among other things) maintains a pointer to the last node of the list as shown below:

```
void printList(nodeType* head);
// This function takes the head pointer to a list as its argument, and
// systematically prints the integers appearing on the list.

int length(nodeType* head);
// This function takes the head pointer to a list as its argument, and
// returns the length of the list, i.e., number of elements on the
// list.

bool isPresent(nodeType* head, int target);
// This function determines whether a particular integer appears
// on the list.
// In: Head pointer of the list and the integer being searched
// Out: true if the given integer appears on the list and
//      false, otherwise.
```

In the alternative implementation, the functions `printList`, `length` and `isPresent` are realized recursively.