

Fundamentals of Computing: Lecture 26

Piyush P Kurur
Office no: 224
Dept. of Comp. Sci. and Engg.
IIT Kanpur

October 7, 2009

Recursive data types

- ▶ List.

Recursive data types

- ▶ List. A list is either an empty list or an element followed by a list.

Recursive data types

- ▶ List. A list is either an empty list or an element followed by a list.
- ▶ Binary trees.

Recursive data types

- ▶ List. A list is either an empty list or an element followed by a list.
- ▶ Binary trees. A binary tree is either an empty Tree or a root node with two children left subtree and right subtree.

Recursive data types

- ▶ List. A list is either an empty list or an element followed by a list.
- ▶ Binary trees. A binary tree is either an empty Tree or a root node with two children left subtree and right subtree.
- ▶ General trees (some times called Rose trees).

Recursive data types

- ▶ List. A list is either an empty list or an element followed by a list.
- ▶ Binary trees. A binary tree is either an empty Tree or a root node with two children left subtree and right subtree.
- ▶ General trees (some times called Rose trees). Either an empty tree or a node with a Forest of subtrees.

List

```
data List a = Empty | Cons a (List a)
```


List

```
data List    a = Empty | Cons a (List a)

typedef struct Cons Cons;
typedef Cons *List
struct Cons {
    int datum;
    List next;
};
List emptyList = (List) NULL;
```

```
int head(List l)
{
    if( l == NULL) {error("head of an empty list");}
    else return l -> datum
}
List tail (List l)
{
    if( l == NULL) {error("tail of an empty list");}
    else return l -> next
}
```

Some list functions

Function `singleton(x)` creates a list of just one element.

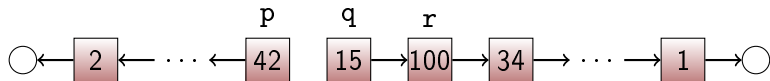
Some list functions

Function `singleton(x)` creates a list of just one element.

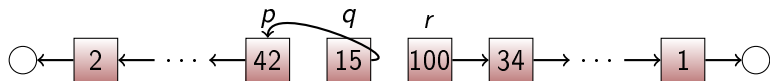
```
List singleton(int x){
    List l;
    l = (List) malloc(sizeof(Cons))
    if( l != NULL)
    {
        l -> datum = x;
        l -> next  = NULL;
    }
    return l;
}
```

```
void appendTo(List *a, List b)
{
    List ptr;
    if( *a == NULL){
        *a = b;
        return;
    }
    ptr = *a;
    while(ptr -> next != NULL)
    {
        ptr = ptr -> next;
    }
    ptr -> next = b;
    return;
}
```

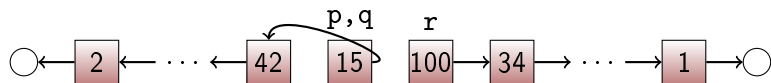
Reverse a list



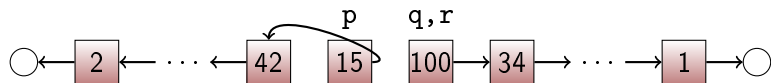
Reverse a list



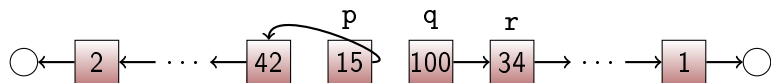
Reverse a list



Reverse a list



Reverse a list



Reverse a list



```
void reverse(List a)
{
    List p,q,r;
    if( a == NULL) return;
    p = NULL;
    q = a;
    r = a -> next;
    while( r )
    {
        q -> next = p;
        p = q;
        q = r;
        r = r -> next;
    }
}
```