ADT Liste semplici

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2012/2013

Section 1

Liste semplici: definizione

Liste semplici

- Una lista è un Abstract Data Type
- Rappresenta una collezione ordinata di elementi dello stesso tipo
- Il numero di elementi della collezione è finito ma (teoricamente) illimitato
- Differenze dagli array:
  - Dimensione non predefinita
  - Primitive di accesso diverse

Liste semplici: definizione ricorsiva

Una lista di elementi di tipo `T` è:

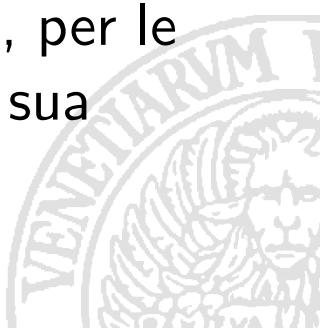
- la lista vuota (NULL)
- Un elemento di tipo `T` (chiamato *testa*) seguito da una lista di tipo `T` chiamata *coda*

Questa definizione *ricorsiva* suggerisce che molti problemi sulle liste si prestano ad ammettere una soluzione ricorsiva

Primitive sulle liste (mutabili)

- Costruttore di liste vuote `new_list`
- Test di lista vuota `is_empty`
- Inserimento di un elemento in test `prepend`
- Inserimento di un elemento in coda `append`
- Acquisizione dell'elemento di testa su lista non vuota `head`
- Acquisizione della coda di una lista non vuota `tail`

Da notare che a differenza di quello che avviene per i vettori, per le liste non definiamo l'accesso diretto ad un elemento data la sua posizione



Pseudo-codice per determinare la lunghezza di una lista

Data la lista L

- Se L è la lista vuota, allora la lunghezza è 0
- Se L non è la lista vuota allora è formata da una testa H e una coda C. Quindi la lista ha elementi $1+$ numero di elementi di C

Esempi di operazioni implementabili sulle liste

- Confrontare due liste per determinare se contengono elementi con lo stesso valore e nello stesso ordine
- Accedere un elemento che si trova in una determinata posizione
- Inserire un nuovo elemento in una certa posizione della lista
- Rimuovere un elemento che si trova ad una particolare posizione nella lista

Section 2

Implementazione delle liste

Come implementare le liste?

- Non dobbiamo confondere l'implementazione dell'ADT con la sua definzione
- In generale un ADT ha più implementazioni possibili
- Alcuni linguaggi (es. F#) hanno le liste come tipo predefinito
- L'implementazione più comune delle liste in C fa uso massiccio di puntatori
  - Faremo riferimento a questa implementazione

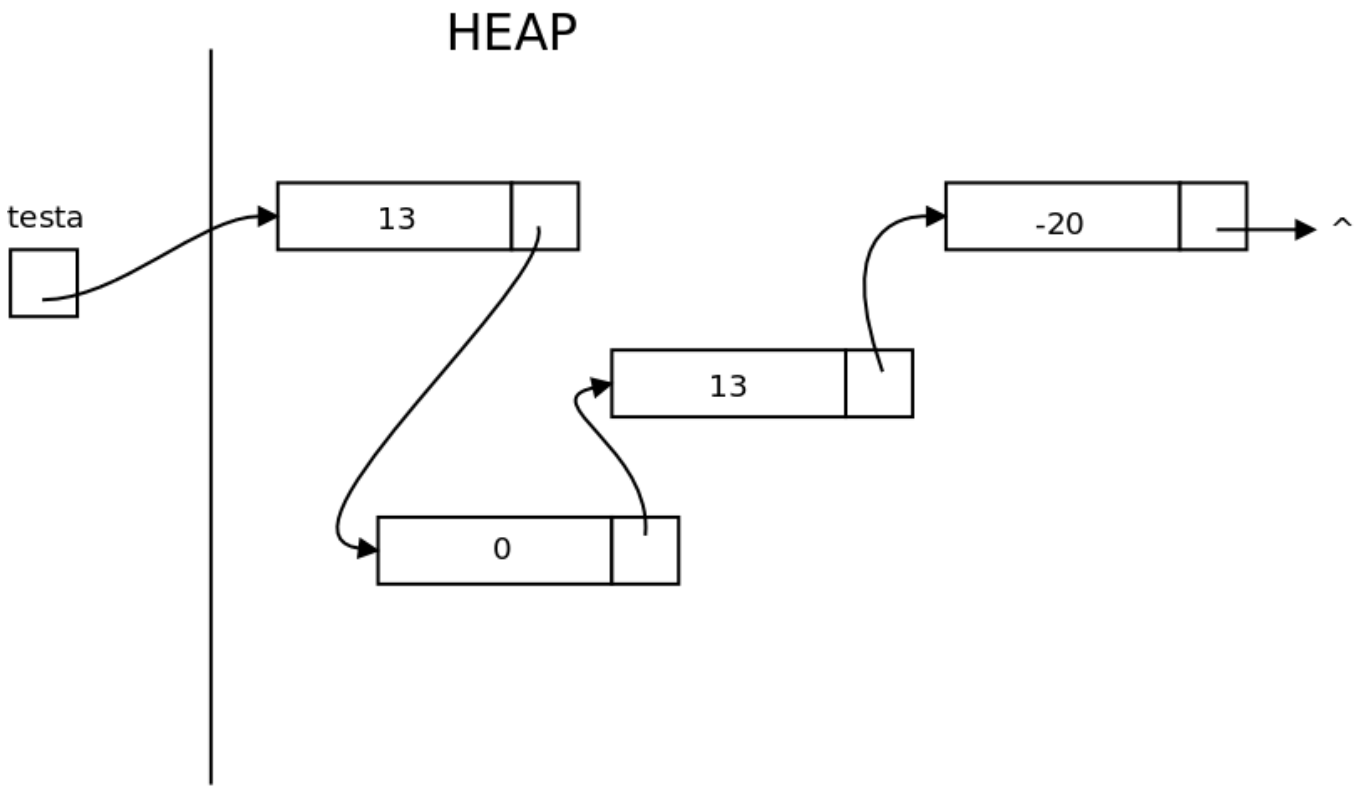
Definizione del tipo

```
struct listint {
    int info;
    struct listint* next;
};

typedef struct listint* Listint;
```

- Osservare la strutturazione ricorsiva nella definizione del tipo

Esempio



Inserimento in testa

```
int prepend(Listint* l, int elem) {
    Listint newcell;
    newcell = (Listint) malloc(sizeof(struct listint));
    if (newcell) {    /*Inserimento con successo*/
        newcell->next = *l;
        newcell->info = elem;
        *l = newcell;
        return 1;
    }
    else   /*Inserimento fallito*/
        return 0;

}
```

Inserimento in coda (ricorsivo)

```
int append(Listint* l, int elem) {
    if (*l==NULL) { /*lista vuota*/
        *l = (Listint) malloc(sizeof(struct listint));
        if (*l) {  /*Inserimento con successo*/
            (*l)->next = NULL;
            (*l)->info = elem;
            return 1;
        } else  /*Inserimento fallito*/
            return 0;
    }
    else /*lista non vuota*/
        return append(&((*l)->next), elem);
}
```

Inserimento in coda (iterativo)

```
int append_iter(Listint*  l, int elem) {
    Listint newcell;
    newcell = (Listint) malloc(sizeof (struct listint));
    if (newcell) {/*allocazione corretta*/
        newcell->info = elem;
        newcell->next = NULL;

        if (!(*l))
            *l = newcell;
        else {
            Listint pc = *l;
            while (pc->next) /*scorre la lista*/
                pc = pc->next;
            pc->next = newcell;
        }
        return 1;
    }
    else
        return 0;
}
```

Distruzione della lista (ricorsiva)

- L'algoritmo distrugge la lista a partire dalla coda

```
void destroy(Listint mylist) {
    if (mylist) {
        destroy(mylist->next);
        free(mylist);
    }
}
```

Distruzione della lista (iterativa)

```
void destroy_iter(Listint mylist) {
    Listint pc = mylist;
    while (pc) {
        mylist = pc->next;
        free(pc);
        pc = mylist;
    }
}
```

- Con che ordine avviene l'eleminazione delle celle nella versione ricorsiva e nella versione iterativa?

Lunghezza della lista (ricorsiva vs. iterativa)

```
int length_rec(Listint l) {
    if (l)
        return 1 + length_rec(l->next);
    else
        return 0;
}

int length_iter(Listint l) {
    int lung=0;
    while (l) {
        lung++;
        l = l->next;
    }
    return lung;
}
```

Esercizi

- Scrivere una funzione che decida se una lista di interi contiene solo numeri pari. Fornire una soluzione iterativa e una ricorsiva
- Scrivere una funzione che decida se due liste di interi sono uguali. Fornire una soluzione iterativa e una ricorsiva
- Scrivere una funzione che date due liste `l1` e `l2` lasci `l2` inalterata, e `l1` venga modificata concatenandola con `l2` (fare attenzione al tipo dei parametri formali)
- Scrivere una funzione che data una lista di interi e un intero e restituisca la posizione della prima occorrenza di e nella lista. La prima cella ha posizione 0. Nel caso l'elemento non sia presente la funzione ritorna -1. Fornire una versione ricorsiva e una iterativa.