SAPIENZA
UNIVERSITÀ DI ROMA

Ingegneria Informatica e Automatica

Tecniche di Programmazione

A.A. 2016/2017

Tipi di dato astratti

Dato astratto

- Non dipende dal linguaggio di programmazione.
- E' costituito da tre componenti:
  - Il dominio di interesse, che rappresenta l'insieme di valori che costituiscono il tipo astratto;
  - Un insieme di costanti, che denotano valori del dominio di interesse
  - Un insieme di **operazioni** che si effettuano sugli elementi del dominio di interesse, utilizzando eventualmente elementi degli altri domini.

Esempio: il tipo astratto Booleano

- Dominio di interesse:
  - { vero, falso }
- Costanti:
  - **true** e **false**, che denotano rispettivamente vero e falso
- Operazioni:
  - and, or, not, ecc...

Tipo booleano in C del tipo

- Dominio:
  - il valore falso si rappresenta con il valore 0.
  - il valore vero si rappresenta tutti il valore 1.
- Costanti:
  - typedef int bool;
    #define TRUE 1
    #define FALSE 0
- Operazioni: si possono utilizzare direttamente le operazioni del C:
  - &&, ||, !

Osservazioni

- Per uno stesso tipo astratto si possono avere più implementazioni diverse.
- Ogni implementazioni può avere aspetti positivi e negativi.
  - E’ necessario valutare il campo di applicazione per decidere quale implementazione usare
- Un tipo di dato può essere parametrico, cioè definito a partire da uno o più tipi di dato (struttura di dati).

Esempio: tipo di dato lista

- Una lista è una sequenza di dati omogenei.
- Dominio:
  - tutte le sequenze di elementi del tipo contenuto nella lista.
- Costanti:
  - Dipende dall'implementazione
- Operazioni:
  - Restituisce il primo elemento della lista, aggiunge un elemento in testa, elimina l'ultimo elemento, ecc...

Implementazione 1 del tipo lista

- Si vuole implementare un tipo lista con **Linked List**:

```
typedef struct
{
    …;
}Data;

typedef struct ListNode ListNode;
struct ListNode
{
    Data info;
    ListNode *next;
};

typedef struct
{
    ListNode *head;
} List;
```

Implementazione 1 del tipo lista

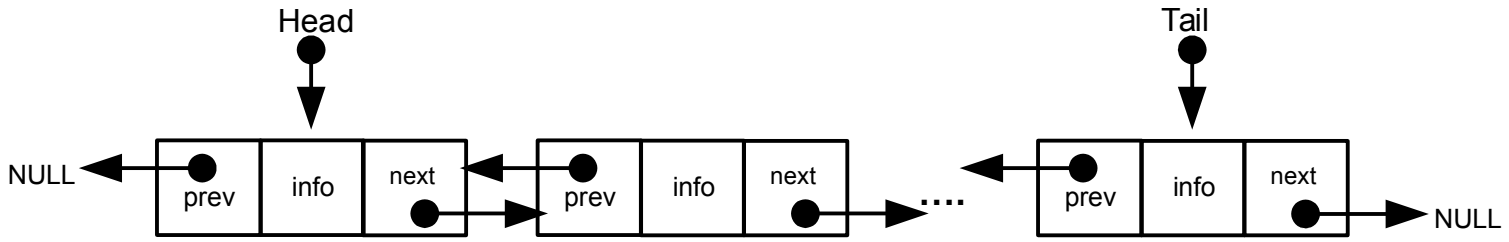
- Operazioni:

```
// Restituisce true se la lista vuota
bool emptyLista(List lis);
// Restituisce il primo elemento di una lista non vuota
void firstElem(List lis, ListNode *e);
// Restituisce l'ultimo elemento di una lista non vuota
void lastElem(List lis, ListNode *e);
// Modifica la lista togliendo il primo elemento se non vuota
void deleteFirst(List *list);
// Modifica la lista togliendo l'ultimo elemento se non vuota
void deleteLast(List *list);
// Inizializza la lista assegna al parametro la lista vuota
void initLista(List *list);
// Modifica la lista aggiungendo l'elemento e in prima posizione
void addElem(List *list, ListNode e);
// Stampa a schermo la lista
void printList(List list );
// ...
```

Possibile soluzione → complementi_pretto/list1

Implementazione 2 del tipo lista

Si vuole modificare l'implementazione precedente utilizzando una lista a link doppio (**doubly linked list**):




```
typedef struct ListNode ListNode;
struct ListNode
{
     Data data;
     ListNode *next, *prev;
};

typedef struct
{
     ListNode *head, *tail;
}List;
```

Possibile soluzione → complementi_pretto/list2

Implementazione 3 del tipo lista

- Una implementazione del tipo di dato lista si può ottenere anche tramite array.

```
typedef struct
{

     Data *info;
     Int size;

}List;
```

Il tipo di dato pila

- Una pila (**stack**) è una sequenza di elementi (tutti dello stesso tipo) in cui l'inserimento e l'eliminazione di elementi avvengono secondo la regola LIFO ("*Last In, First Out*").
- Le classiche operazioni su una pila sono:
  - top: restituisce l'elemento in cima alla pila
  - pop: elimina primo elemento
  - push: aggiunge un elemento alla pila

Implementazione 1 del tipo pila

- Si vuole rappresentare la pila mediante una linked list:

```
typedef struct
{
     …;
}Data;

typedef struct StackNode StackNode;
struct StackNode
{
     Data info;
     StackNode *next;
};

typedef struct
{
     StackNode *head;
} Stack;
```

Implementazione 1 del tipo pila

- Operazioni:

```
// Restituisce true se la pila vuota
bool emptyStack(Stack Stack);
// Inizializza una pila vuota
void initStack(Stack *Stack);
// aggiunge un elemento alla pila
void push(Stack *Stack, StackNode *e);
// Restituisce l'elemento in cima alla pila
StackNode *top(Stack Stack );
// Elimina primo elemento
void pop(Stack *Stack);
```

Possibile soluzione → complementi_pretto/stack1

Implementazione 2 del tipo pila

- Si vuole rappresentare la pila mediante un array:

```
#define STACK_MAX_SIZE 100

typedef struct
{
    …;
}Data;

typedef struct
{
    Data data[STACK_MAX_SIZE];
    int pos;
} Stack;
```

Possibile soluzione → complementi_pretto/stack2

Implementazione 3 del tipo pila

- Per utilizzare un vettore senza dover fissare la dimensione massima della pila a tempo di compilazione si può utilizzare un vettore allocato dinamicamente:
  - Si fissa una dimensione iniziale del vettore (che determina anche quella minima)
  - La dimensione può crescere a seconda delle necessità

Possibile soluzione → complementi_pretto/stack3

Il tipo di dato coda

- Una pila (**queue**) è una sequenza di elementi (tutti dello stesso tipo) in cui l'inserimento e l'eliminazione di elementi avvengono secondo la regola FIFO ("*First In, First Out*").
- Le classiche operazioni su una coda sono:
  - enqueue: inserisce un elemento nella testa della queue.
  - dequeue: restituisce e rimuove l'elemento elemento nella coda della lista.

Implementazione 1 del tipo coda

- Si vuole rappresentare la coda mediante un array:

```
#define QUEUE_MAX_SIZE 100

typedef struct
{
    …;
}Data;

typedef struct
{
    Data data[QUEUE_MAX_SIZE];
    int first, last;
} Queue;
```

Implementazione 1 del tipo coda

- Operazioni:

```
// Inizializza una coda vuota
void initQueue( Queue *q );
// Restituisce TRUE se la coda è vuota
bool emptyQueue( Queue *q );
// Restituisce TRUE se la coda è piena
bool fullQueue( Queue *q );
// Inserisce un elemento in testa
void enqueue( Queue *q, Data d);
// Estrae un elemento in coda
void dequeue( Queue *q, Data *d);
```

Possibile soluzione → complementi_pretto/queue1

Implementazione 2 del tipo coda

- Una implementazione del tipo di dato coda si può ottenere anche tramite una lista a link singolo:

```
typedef struct QueueNode QueueNode;
struct QueueNode
{
     Data data;
     QueueNode *next;
};

typedef struct
{
     QueueNode *first, *last;
}Queue;
```