Tipi di Dato (TDD)

Cosa Sono:

Un **tipo** è una collezione di valori.

int, bool, char […] sono esempi di **tipo semplice** mentre un insieme composto da nome, indirizzo, codice fiscale […] è un **tipo aggregato o composto**.

Formalizzazione: un tipo è un insieme

Un **dato** è un elemento il cui valore è preso da un tipo. (Ad esempio 1 è un dato che tipo intero. Si dice che 1 è un membro o elemento del tipo intero.)

Formalizzazione: un dato è un elemento di un insieme.

Un **tipo di dato** (concreto) **TDD,** è un tipo con una collezione di operazioni per manipolarne gli elementi (o membri). **Più formalmente** è un'algebra, solitamente eterogenea, **su una segnatura**. Una **segnatura** è una coppia **(S,O)**, con **S** insieme con elementi definiti sort, e **O** famiglia di operatori su queste sort. Es. int, stack e push, pop per uno stack di interi

Un **tipo di dato astratto** è una classe di algebre, solitamente eterogenee, sulla stessa segnatura. (anche se non è vero, facciamo finta che “Tipo di dato” e “Tipo di dato astratto” siano sinonimi e semplifichiamoci la vita...)

N.B *Useremo il termine struttura dati riferendoci a una particolare organizzazione delle informazioni che permette di supportare in modo efficiente le operazioni di un tipo di dato*

SINTESI

Interfaccia del tipo di dato =

(formalmente) segnatura many-sorted, ovvero eterogenea.

(informalmente) = insieme di “sort” o tipi e famiglia di operatori su queste sort, per es. Int, stack e push, pop per uno stack di interi.

Tipo di dato =

(formalmente) algebra su una segnatura eterogenea

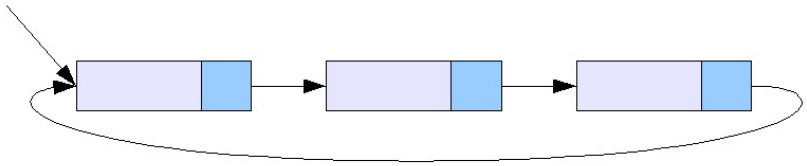
(informalmente) = la segnatura dell'algebra + una specifica astratta del comportamento degli operatori: degli assiomi, ma anche una descrizione informale

Perche I TDD sono cosi importanti

I TDD specificano l'interfaccia (sintassi) e il significato che le operazioni nell'interfaccia devono assumere (semantica), consentendo di realizzare il principio dell'incapsulamento o **information hiding**.

Elenco dei TDD e descrizione di essi

- **Liste (**o "successioni finite")



Algebra eterogenea in cui le sort sono List, N, Bool, Elem (dove Elem è una sort non ulteriormente specificata) e le operazioni, con loro proprietà, sono:

- emptyList → List
- set N x Elem x List → List
- add N x Elem x List → List
- addBack Elem x List → List
- addFront Elem x List → List
- removePos N x List → List
- get: N x List → Elem
- isEmpty List → Bool
- size List → N

N.B:

"N x List" *specifica gli argomenti che l'operazione accetta. In questo caso, "N" rappresenta un numero intero e "List" rappresenta una lista. Quindi, "removePos" accetta due argomenti: un numero intero che rappresenta la posizione dell'elemento da rimuovere e una lista dalla quale rimuovere l'elemento.*

N.B

- **List**: è il tipo di dato che rappresenta una lista. È una sequenza di elementi di un tipo non specificato chiamato "Elem".
- **N**: è un tipo di dato che rappresenta un numero intero (positivo o negativo).
- **Bool**: è un tipo di dato che può avere solo due valori: vero (true) o falso (false).

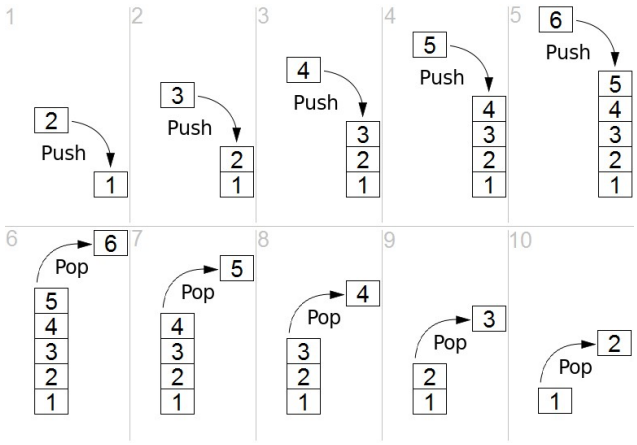
- **Elem**: è un tipo di dato che rappresenta gli elementi all'interno della lista.

Le operazioni che possono essere eseguite sulla lista sono:

- **emptyList**: Restituisce una lista vuota.
- **set**: Aggiunge un elemento alla lista in una posizione specifica.
- **add**: Aggiunge un elemento alla lista in una posizione specifica.
- **addBack**: Aggiunge un elemento alla fine (coda) della lista.
- **addFront**: Aggiunge un elemento all'inizio (testa) della lista.
- **removePos**: Rimuove un elemento dalla lista in una posizione specifica.
- **get**: Restituisce l'elemento dalla lista in una posizione specifica.
- **isEmpty**: Verifica se la lista è vuota o meno, restituendo vero se è vuota e falso altrimenti.
- **size**: Restituisce il numero di elementi nella lista.

Le liste (“List”) sono un TDD, ma indichiamo con la parola “lista” anche la famiglia di strutture dati che si ottengono collegando struct mediante puntatori: liste semplici, liste doppiamente collegate, liste doppiamente collegate circolari, liste doppiamente collegate circolari con sentinella. Un'altra fonte di ambiguità a cui prestare attenzione...

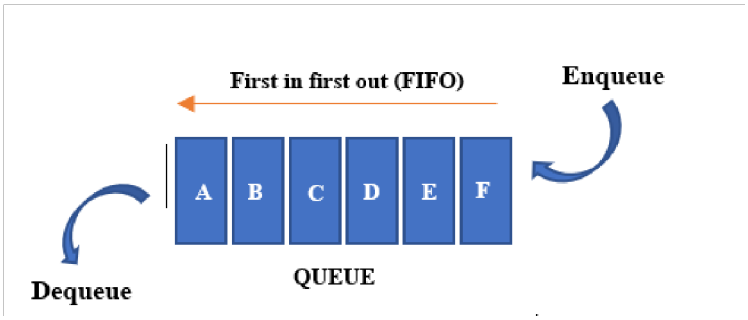
- **Pile (Stack)**



E' una sequenza di elementi di un certo tipo, in cui è possibile aggiungere o togliere elementi soltanto ad un estremo (la testa della sequenza). Una pila può essere quindi vista come un caso speciale di **lista** in cui l'ultimo elemento è anche il primo ad essere rimosso e non è possibile accedere ad alcun elemento che non sia quello in testa.

- isEmpty() → result
- push(elem e)
- pop() → elem
- top() → elem

- **Code (Queue)**



E' una sequenza di elementi di un certo tipo, in cui è possibile aggiungere elementi ad un estremo e toglierne dall'estremo opposto. Una coda può essere quindi vista come una particolare lista in cui è possibile aggiungere elementi ad un estremo (il fondo) e togliere elementi dall'altro estremo (in testa).

- isEmpty() → result restituisce true se S è vuota
- enqueue(elem e) aggiunge e come ultimo elemento di S

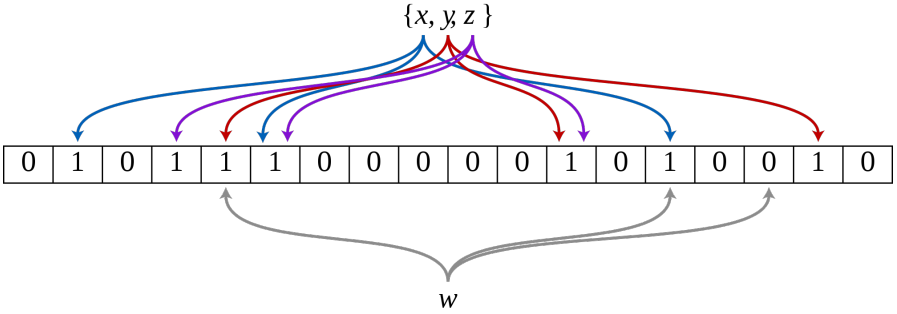
- dequeue() → elem toglie da S il primo elemento e lo restituisce
- first() → elem restituisce il primo elemento di S

- **Insiemi (Set)**

  Sono un aggregato di elementi a valori omogenei (provenienti dallo stesso dominio) e distinti (**senza ripetizioni**).

  - emptySet → Set
  - insertElem Elem x Set → Set
  - deleteElem Elem x Set → Set
  - setUnion Set x Set → Set
  - setIntersection Set x Set → Set
  - setDifference Set x Set → Set
  - isEmpty Set → Bool
  - isSubset Set x Set → Bool
  - size Set → N
  - member Elem xSet → Bool

- **BitVector**



Le operazioni degli operatori bit a bit servono a controllare, modificare o spostare i bit di un valore binario. Danno il risultato in base ai bit del/i valore/i.

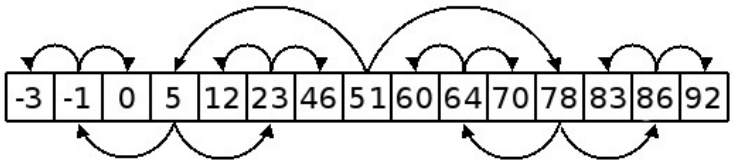
Bit vector e operazioni &(and), ~(not) ed | (or)

- **"&" (AND):** Questo operatore confronta i bit corrispondenti in due operandi e restituisce un risultato in cui ogni bit è 1 se e solo se entrambi i bit corrispondenti nei due operandi sono 1. Ad esempio, se hai 1010 (10 in decimale) e 1100 (12 in decimale), il risultato dell'operazione "&" sarà 1000 (8 in decimale).
- **"~" (NOT):** Questo operatore inverte tutti i bit in un numero, trasformando ogni 0 in 1 e ogni 1 in 0. Ad esempio, se hai 1010 (10 in decimale), il risultato dell'operazione "~" sarà 0101 (5 in decimale).
- **"|" (OR):** Questo operatore confronta i bit corrispondenti in due operandi e restituisce un risultato in cui ogni bit è 1 se almeno uno dei bit corrispondenti

nei due operandi è 1. Ad esempio, se hai 1010 (10 in decimale) e 1100 (12 in decimale), il risultato dell'operazione "|" sarà 1110 (14 in decimale).

```
unsigned int x1, x2;
    x1 = 0;
    x2 = ~0;                              //  tutti i bit impostati a 1
    cout << "0: " << x1 << endl;     // Stampa il valore di x1 (0)
    cout << "~0: " << x2 << endl;    // Stampa il valore di x2 (~0, ovvero tutti i bit impostati a 1)
    cout << "~~0: " << ~x2 << endl;  // Stampa il valore ottenuto invertendo nuovamente i bit di x2 (0)
    x1 = 2;
    x2 = 4;
    cout << x1 << " | " <<  x2 << " = " << (x1|x2) << endl;  // Stampa il risultato dell'operazione OR tra x1 e x2
    x1 = 2;
    x2 = 4;
    cout << x1 << " & " << x2 << " = " << (x1 & x) << endl;
    x1 = 6;
    x2 = 4;
    cout << x1 << " | " << x2 << " = " << (x1 | x) << endl;
    x1 = 6;
    x2 = 4;
    cout << x1 << " & " << x2 << " = " << (x1 & x) << endl;
```

- **Array Ordinato**



Un'altra struttura dati che possiamo usare, è un array (con size e maxsize) che manteniamo **ordinato**.

L'ordinamento rende meno efficienti alcune operazioni, ma più efficiente la ricerca, l'unione e l'intersezione.

- **Dizionario**

  - insert (elem e, chiave k) aggiunge a S una nuova coppia (e,k) ○ delete(chiave k) cancella da S la coppia con chiave k.
  - search (chiave k) -> elem se la chiave k è presente in S restituisce l'elemento ad esso associato, altrimenti null.

- **Tabelle Hash (strutture dati)**

Sono dizionari basati sulla proprietà di accesso diretto alle celle di un array

- Chiavi prese da un universo totalmente ordinato U (possono non essere numeri)
- Funzione hash: h: U → [0, m-1] (funzione che trasforma chiavi in indici)
- Elemento con chiave k in posizione v[h(k)] ideali come supporto per TDD

dizionario e TDD insieme. ● **Albero (Tree)**