ADT

Abstract data type

ADT

- Un tipo di dato astratto è una terna <S,F,C> dove
  - s = domini di interesse
  - f = insieme di funzioni
  - C = insieme di costanti

Esempio: tipo insieme

- S = {boolean, tipoelementi, insieme}
- F = {$\cup$: insieme x insieme $\rightarrow$insieme

  $\cap$: insieme x insieme $\rightarrow$insieme

  $\in$: tipoelementi x insieme $\rightarrow$boolean

  null : insieme $\rightarrow$boolean

  }
- C = $\varnothing$

Definizione del tipo astratto insieme cioe' dei nomi
dei tipi, delle funzioni e delle costanti

Implementazione della libreria insieme.h contenente la definizione
dei tipi in S, delle funzioni in F e delle costanti dell’insieme C

Uso del tipo di dato astratto:

#include insieme.h

Liste semplici

Il tipo lista semplice è un ADT

<S,F,C>

dove

S = { lista , atomo, boolean}

F = { cons, car, cdr, null:

cons: lista x atomo $\rightarrow$lista

car: lista $\rightarrow$atomo

cdr: lista $\rightarrow$lista

null: lista $\rightarrow$boolean}

C = { lista_vuota} dove lista_vuota è la costante che denota la lista che non contiene elementi

Liste semplici

- Il tipo astratto lista consente di rappresentare sequenze di elementi di un determinato tipo (atomo)
- Per sequenza si intende un insieme finito e ordinato di elementi
- Esempio (notazione parametrica)
  - ( ) lista vuota
  - (8, 25, 6, 90, 6)
  - (52)

Funzioni e liste

- cdr applicata alla lista (8, 25, 6, 90, 6) ritorna la lista (25, 6, 90, 6)
- car applicata alla lista (8, 25, 6, 90, 6) ritorna 8
- cons applicata alla lista (8, 25, 6, 90, 6) e al valore 9 ritorna la lista (9, 8, 25, 6, 90, 6)
- Definizione ricorsiva di lista
  - **Ogni valore di tipo lista è o la lista_vuota o un valore del tipo atomo seguito da una lista**

Liste mediante Array

```
#include “stdio.h”

typedef int TAtomo;
typedef struct elem {
    int          testa, numEl, maxEl;
    TAtomo   *e;
} TLista;
```

Liste mediante Array

```
int InizializzaLista (Tlista *PL, int N ) {
    PL->e = (Tatomo *) malloc(sizeof(TAtomo)*N);
    if (PL->e == NULL) return 0;
    PL->numEl = 0;
    PL->testa = -1;
    return PL -> maxEl = N;
}

int null (TLista L) {
    return L.numEl == 0;
}
```

Lista Mediante Array (cont.)

```
int cdr ( TLista *PL ) {
    if (null(*PL)) return -1;
    if ( PL->numEl == 1 ) PL->testa = -1
    else PL->testa--;
    return --(PL->numEl);
}

TAtomo car( TLista L ) {
    if( null(L) ) return ERR;
    return L.e[L.testa];
}
```

Lista Mediante Array (cont.)

```
int cons( TLista *PL, TAtomo A ) {
    if( PL->numEl == PL->maxEl) return -1;
    PL->testa++
    PL->e[PL->testa] = A;
    return ++(PL -> numEl);
}

int cancella_lista( TLista *PL) {
    free( PL->e);
    PL->e = NULL;
}
```

Lista Semplice Collegata mediante array

```
#include “stdio.h”
#define ERR   -9999

typedef     int TAtomo;
typedef struct StMatrice {
    TAtomo      dato;
    int         succ;
} TMatrice;
typedef struct StLista {
    TMatrice    *e;
    int         NMax;
    int         primo, libera;
} Tlista, *PTLista;
```

Lista Semplice Collegata

```
void InizializzaListaLibera( TLista *PL ) {
    int   p;
    for( p = 0; p < (PL->NMax - 1); p++ ) {
     PL->e[p].succ = p+1;
    }
    PL->e[NMax - 1].succ = -1;
    PL ->libera = 0;
}
```

```
int InizializzaLista( Tlista *PL, int N ) {
    PL->e = (Tmatrice *)malloc(sizeof(TMatrice)*N);
    if( PL->e  == 0 ) return -1;
    PL->primo = -1;
    InizializzaListaLibera( PL );
    return PL->NMax = N;
}
```

```
int null( TLista L ) {
    return L.primo == -1;
}

int cdr ( TLista *PL ) {
    int   temp;
    if( null(*PL) ) return -1;
    temp = PL->primo;
    PL->primo = PL->e[PL->primo].succ;
    PL->e[temp].succ = PL->libera;
    PL->libera = temp;
    return 1;
}
```

```
int cons( PTLista PL, TAtomo A) {
    int   temp;
    if ( PL->libera == -1 ) return -1;
    temp = PL->libera;
    PL->libera = PL->e[temp].succ;
    PL->e[temp].succ = PL->primo;
    PL->primo = temp;
    PL->e[PL->primo].dato = A;
    return  1;
}
```

```
TAtomo car( TLista PL ) {
    return null( PL ) ? ERR : PL.e[PL.primo].dato;
}
```

Lista Semplice Collegata mediante puntatori

```
#define ERR   -9999
typedef intTAtomo;
typedef struct StLista {
    TAtomo      dato;
    struct StLista    * succ;
} Elista, *TLista;
```

TAtomo car (TLista L);

int null (TLista L);

*void cons(TLista *PL, TAtomo A) ;*

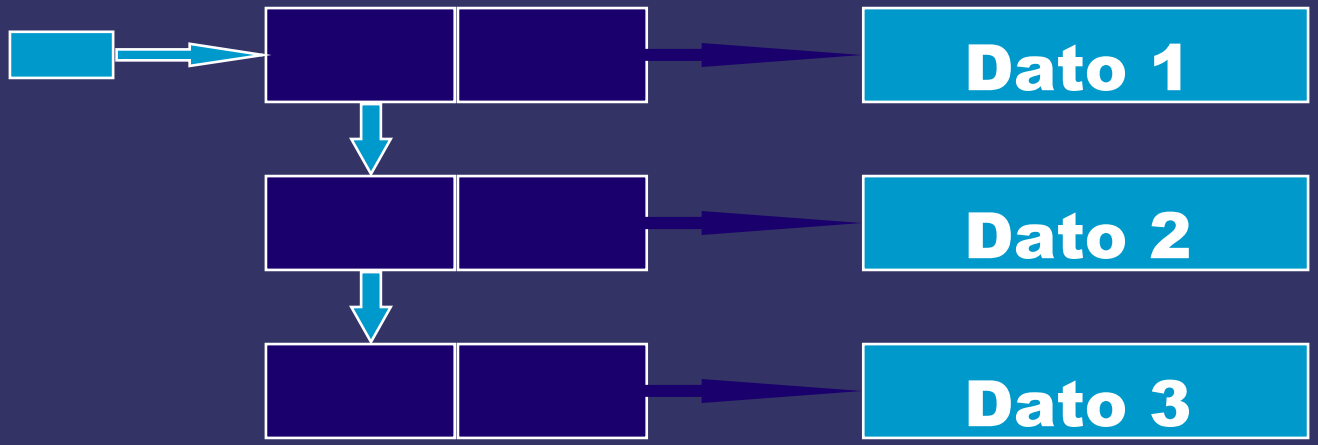
*int cdr(TLista *PL);*

```
int null( TLista L ) {
    return L = = NULL;
}

TAtomo car( TLista L ) {
    return null(L) ? ERR :L->dato;
}
```

```
int cdr( TLista *PL ) {
    TLista   temp;
    if ( null(*PL) ) return 0;
    temp = *PL;
    *PL = (*PL)->succ;
    free(temp);
    return 1;
}
```

```
void cons( TLista *PL, TAtomo A ) {
    TLista   temp;
    temp = (TLista) malloc( sizeof(ELista) );
    if (temp = = NULL) return
    temp->dato = A;
    temp->succ = *PL;
    *PL = temp;
}
```



Utilizzando il C e' possibile creare una lista *generica*, cioe' in cui posso memorizzare atomi di qualsiasi tipo.

Liste ordinate

inserimento nella lista vuota o inserimento del valore 2 nella lista L equivale ad un inserimento in testa

inserimento del valore 6 nella lista L:

1. cerca la posizione dove inserire (è necessario il puntatore all'elemento precedente)
2. alloca la memoria per un nuovo elemento
3. collega il nuovo elemento

Liste ordinate

```
typedef int       TAtomo;
typedef struct StLista {
          TAtomo            dato;
          struct StLista  * succ;
} ELista, *TLista;

void insord(TLista *P, TAtomo T)
{if (null(*P) || (car(*P)>T)) cons(P,T);
          else  insord(&((*P)->succ),T)
}
```

Liste ordinate

```
int canc(TLista *P, TAtomo T)
{    if (null(*P) || car(*P)>T) return -1;
     if (car(*P) == T) return cdr(P);
     return(canc(&((*P)->succ)),T);
}
```

Liste ordinate (cont.)

```
void insord(TLista *P,  TAtomo T)
{ TLista Q, Prec;
    if (null(*P) || car(*P)>T)
        cons(P,T);
    else
    {   Q = *P;
        while (!null(Q) && (car(Q)<T))
        {   Prec=Q;
            Q=Q->succ;
        }
       Q=(TLista)malloc(sizeof(ELista));
        Q->dato = T;
        Q->succ = Prec->succ;
        Prec->succ = Q;
} }
```

Liste ordinate (cont.)

```
int canc(TLista *P,  Tatomo T)
{
    TLista Q, Prec;
    if (null(*P) || (car(*P)>T)) return -1;
    if (car(*P)==T) return cdr(P);
    for (Q=*P; !null(Q)&&(Q->dato<=T); Q = Q->succ)
    {
          if( car(Q) == T )
          {
               Prec->next =Q->next;
           free(Q);
                return 0;
          }
          Prec = Q;
    }
    return -1;
}
```