

Programmazione a oggetti

Abstract Data Types (ADT)

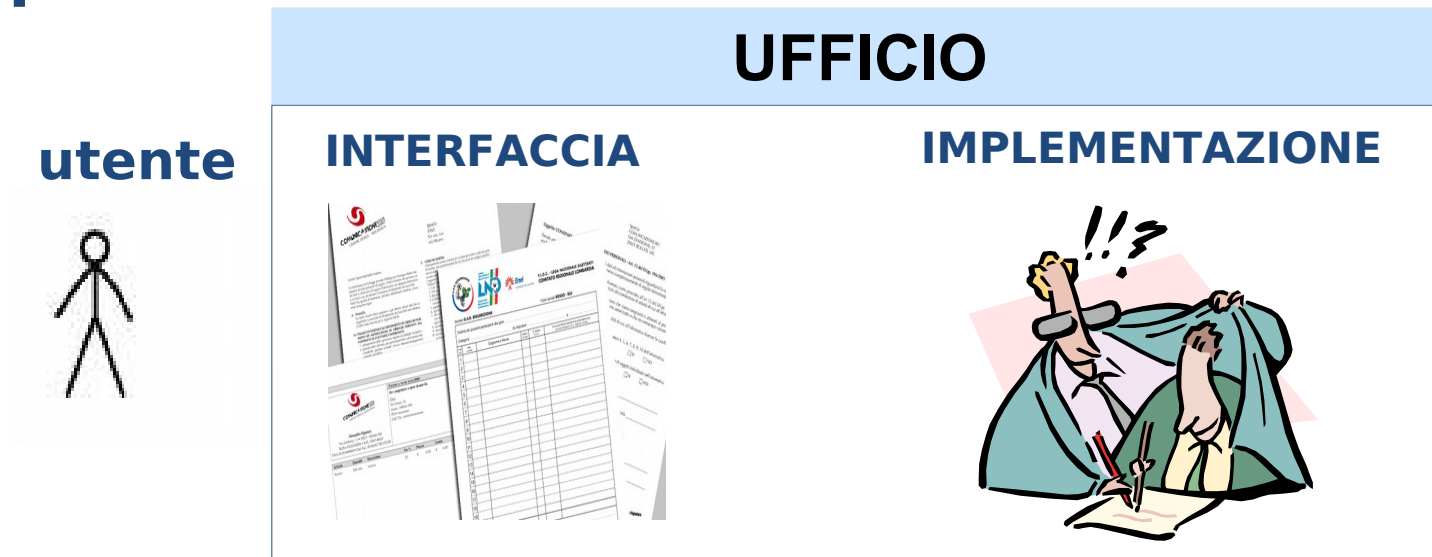
A.A. 2020/2021
Francesco Fontanella

Tipo di Dato Astratto (Abstract Data Type)

- Il concetto di tipo di dato in un linguaggio di programmazione tradizionale è quello di insieme dei valori che può assumere un dato (una variabile).
- I tipi di dato astratto (Abstract Data Type, ADT in inglese) estendono questa definizione, includendo anche l'insieme delle **operazioni possibili** su dati di quel tipo.
- La struttura dati utilizzata per la memorizzazione non è accessibile dall'esterno
 - Si parla di **incapsulamento dei dati** (Information hiding).

Incapsulamento dei dati (Information Hiding)

- Le ADT nascondono i dettagli relativi all'implementazione delle operazioni e le strutture dati utilizzate agli utenti dell'ADT.
- Gli utenti possono usare l'ADT senza conoscere come le operazioni sono implementate
- **Esempio.**



Incaspolamento dei Dati: Vantaggi

- L'utente può usare la ADT solo per mezzo dell'interfaccia definito dallo sviluppatore dell'ADT.
- I dati presenti nell'ADT non possono essere alterati da operazioni scorrette
- L'implementazione della struttura dei dati può essere modificata senza influenzare i moduli (clients) che ne fanno uso

Una esempio di ADT: Insieme

- L'ADT insieme consente di rappresentare collezioni (senza ripetizioni) di elementi di un tipo base
- Operazioni possibili:
 - *insert(x)*
 - *delete(x)*
 - *member(x)*
 - *size()*
 - *empty()*
 - *clear()*

DOMANDE

- Come rappresentiamo l'insieme?
- Con un vettore?
- che tipo di vettore?

Typedef int TipoValue;

La useremo per definire
dati ed operazioni “generiche”,
indipendenti dal tipo di dato



```
typedef struct Set{  
    void init(int l); // funzione di inizializzazione.  
    int size();      // restituisce la cardinalità  
                    // dell'insieme.  
    bool full();     // restituisce TRUE se è pieno.  
    bool empty();    // restituisce TRUE se è vuoto.  
    void clear();    // cancella tutti gli elementi  
    void add(TipoValue el); // aggiunge l'elemento el  
    void del(TipoValue el); // elimina l'elemento el.  
    bool member(TipoValue el); // restituisce TRUE se el è  
                                // presente.
```

private:

```
TipoValue *v;  
int len, n;  
elim(int i); // funzione di eliminazione  
}
```

Qualificatore di accesso.
Rende inaccessibili
all'esterno le variabili e
le funzioni che seguono

ADT Insieme: implementazione



```
void Set::init(int l)
{
    v = new TipoValue[l];
    len = l;
    n = 0;
}
```

Operatore del campo di azione
(scope resolution operator)

```
int Set::size()
{
    return n;
}
```

```
bool Set::full()
{
    return (n == len);
}
```

```
bool Set::empty()  
{  
    return (n == 0);  
}
```

```
bool Set::member(TipoValue val)  
{  
    int i;  
  
    for (i=0; i < n; ++i)  
        if (v[i] == val)  
            return true;  
  
    return false;  
}
```




```
void Set::remove(TipoValue val)
{
    int i;
    bool found;

    i=0;
    found = false;
    while ((i < n) && (!found))
        if (v[i] == val) {
            elim(i);
            found = true;
        } else ++i;

    if (found)
        n--;

    return;
}
```



```
void Set::elim(int pos)
{
    int i;

    for (i=pos; i < n; ++i)
        v[i] = v[i+1];

    return;
}
```

```
void Set::clear()
{
    n = 0;
}

void Set::add(TipoValue val)
{
    if (!full()) {
        if (!member(val))
            v[n++] = val;
    }
    else
        cout<<endl<<"ERRORE: vettore pieno";

}
```

Uso dell ADT Insieme



- Per usare l'ADT appena definita sono necessari tre passi:
 - Associare un tipo a TipoValue;
 - Includere il file set.h;
 - Ricompilare il file set.cpp

Uso dell ADT Insieme: esempio

main.cpp

```
#include <iostream.h>
#include <stdlib.h>

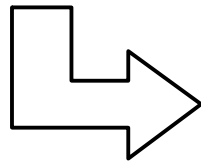
// Assegnazione di tipo
typedef Studente TipoValue;

// inclusione del file
#include "set.h"

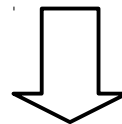
int main(){
    Set appelloEsame;
    .
    .
}
```



COMPILATORE



LINKER



eseguiibile

set.cpp

```
#include <iostream>
#include "studente.h"

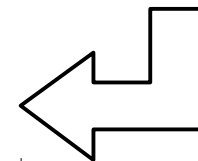
// Assegnazione di tipo
typedef Studente TipoValue;

// inclusione del file
#include "set.h"

void Set::init(int l)
{
    .
    .
    .
}
```



COMPILATORE



NOTA

I file set.h e studente.h contengono le definizioni (specifiche) delle struct set e studente

ADT Pila



- L'accesso ai dati è del tipo **Last In First Out (LIFO)**

PILA	
nome	semantica
init	inizializzazione
push	inserisce un elemento in testa
pop	restituisce e rimuove l'elemento in te
top	restituisce l'elemento in testa
full	la pila è piena?
size	numero di elementi nella pila
empty	la pila è vuota?
clear	cancella i dati contenuti

ADT Pila: specifica



```
struct stack{

    void init(int l);    // funzione di inizializzazione
    bool empty();        // controlla se la pila è vuota
    bool full();         // controlla se la pila è piena

    TipoValue top();    // fornisce l'elemento in testa
    void push(TipoValue val); // inserimento
    TipoValue pop();    // fornisce e rimuove l'elemento in
    testa

private:
    TipoValue *v;       // array per la memorizzazione
    int last;           // punta all'ultimo elemento inserito
    int len;

};
```

ADT Pila: implementazione



```
void stack::init(int l)
{
    v = new TipoValue[l];
    len = l;
    last = -1;

    return;
}
```

```
int stack::size()
{
    return last + 1;
}
```

```
bool stack::full()  
{  
    return (last +1 == len);  
}
```

```
bool stack::empty()  
{  
    return (last == -1);  
}
```



```
void stack::clear()
{
    last = -1;

    return;
}
```

```
TipoValue stack::top()
{
    if (last >= 0)
        return v[last];
    else {
        cout<<"ERRORE: stack vuoto!";
        exit(EXIT_FAILURE);
    }
}
```

```
void stack::push(TipoValue val)
{
    if (last < len)
        v[++last] = val;
    else cout<<"ERRORE: stack pieno";

    return;
}
```

Prima si incrementa l'indice
e poi si inserisce l'elemento

```
TipoValue stack::pop()
{
    if (last >= 0)
        return v[last--];
    else {
        cout<<"ERRORE: stack vuoto!";
        exit(EXIT_FAILURE);
    }
}
```

Prima si copia l'elemento
e poi si decrementa l'indice

ADT Coda

- L'accesso ai dati è del tipo **First In First Out (FIFO)**

CODA	
nome	semantica
init	inizializzazione
add	inserisce un elemento in coda
take	restituisce e rimuove l'elemento in testa
head	restituisce l'elemento in testa
size	Quanto è lunga la coda?
full	la coda è piena?
size	numeri di elementi nella coda
clear	cancella i dati contenuti

ADT Coda: specifica



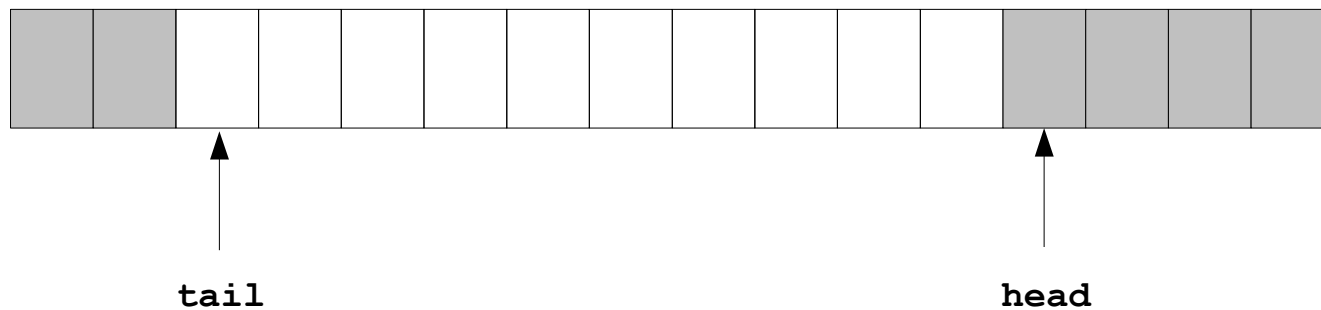
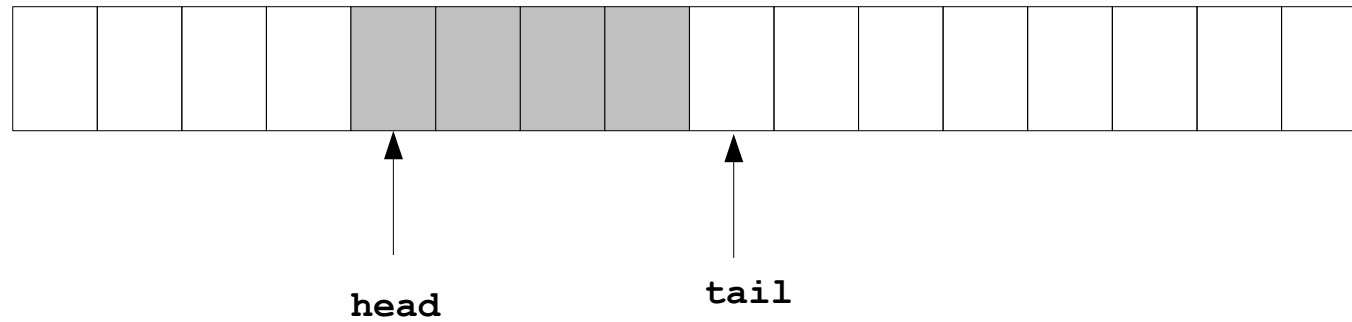
```
struct queue {
```

```
    void init(int l); // funzione di inizializzazione
    int size();        // restituisce l lunghezza della coda
    bool full();       // controlla se la coda è piena
    bool empty();      // controlla se la coda è vuota
    void add(TipoValue val); // aggiunta di un elemento
    TipoValue take();    // estrazione
    TipoValue head();    // restituisce la testa
```

```
private:
```

```
    TipoValue *v; // array di elementi
    int h;         // Punta alla testa della coda
    int t;         // Punta all'ultimo elemento della coda
    int len;       // capacità della coda
};
```

ADT coda: buffer circolare



ADT Coda: implementazione



```
void queue::init(int l)
{
    v = new TipoValue[l];
    h = t = 0;
    len = l;

    return;
}
```

```
void queue::add(TipoValue val)
{
    if (!full())
        v[t] = val;
    else {
        cout<<"ERRORE: coda piena!";
        return;
    }

    // si incrementa la coda.
    t = (t + 1) % len;

    return;
}
```

Incremento in modulo len

```
TipoValue queue::take()
{
    TipoValue tmp;

    If (!empty())
        tmp = v[h]; // Si memorizza la testa
    else {
        cout<<"ERRORE: coda vuota!";
        exit(EXIT_FAILURE);
    }

    // si incrementa la testa.
    h = (h + 1) % len;

    return tmp;
}
```

Incremento in modulo len


```
int queue::size()
{
    if (t >= h)
        return (t - h);
    else return (len - (h - t));
}
```

```
bool queue::full()
{
    return (((t + 1) % len) == h);
}
```

```
bool queue::empty()
{
    return (t == h);
}
```

```
TipoValue queue::head()  
{  
    if (!empty())  
        return v[h];  
    else {  
        cout<<"ERRORE: coda vuota!";  
        exit(EXIT_FAILURE);  
    }  
}
```

ADT Lista



- Una lista è una sequenza finita di elementi:

$$L = \langle a_0, a_1, \dots, a_n \rangle$$

LISTA	
nome	semantica
init	inizializzazione
insert(val, pos)	inserisce il valore val in posizione pos
remove(pos)	rimuove l'elemento in posizione pos
get(pos)	restituisce l'elemento in posizione
set(val, pos)	assegna il valore val all'elemento in posizione pos
size	Quanto elementi ci sono nella lista?
empty	la lista è vuota?
clear	cancella tutti i dati contenuti

ADT Lista: specifica



```
struct List{
    void init();    // funzione di inizializzazione.
    int size();     // restituisce il numero di elementi.
    void clear();   // svuota la lista
    bool empty();   // lista è vuota?
    void insert(TipoValue val, int pos); // inserisce il
        valore val alla posizione pos
    void remove(int pos); // rimuove l'elemento nella
        // posizione pos
    void set(TipoValue val, int pos); // assegna il valore
        // val all'i-esima posizione.
    TipoValue get(int pos); // restituisce il valore
        // dell'elemento alla posizione pos.
private:
    TipoValue *v;
    int n, len;
    void ins(TipoValue val, int pos); // funzione ausiliaria
        // di inserimento
    void elim(int pos); // funzione ausiliaria per
        // l'eliminazione
}
```

ADT Lista: implementazione

```
const int MAX_ELEMENTS = 1000;
```

```
void List::init()  
{  
    len = MAX_ELEMENTS;  
    v = new TipoValue[MAX_ELEMENTS];  
    n=0;  
}
```

```
int List::size()  
{  
    return n;  
}
```

```
void List::clear()  
{  
    n = 0;  
}
```

```
bool List::empty()  
{  
    return (n == len);  
}
```

```
void List::insert(TipoValue val, int pos)
{
    if (pos < 0 || pos > n) {
        cout<<"ERRORE! Valore pos errato!";
        return;
    }

    if (n < len)
        ins(val, pos);
    else {
        cout<<"ERRORE! Lista piena!!";
        return;
    }
    n++; // Si incrementa n.

    return;
}
```

```
Void List::ins(TipoValue val,
               int pos)
{
    int i;

    for (i=pos; i < n; ++i)
        v[i+1] == v[i];

    v[pos] = val;

    return;
}
```

```
void List::remove(int pos)
{
    if (pos < 0 || pos >= n) {
        cout<<"ERRORE! Valore pos errato!";

        return;
    }

    elim(pos);

    return;
}
```



```
void List::set(int pos, TipoValue val)
{
    if (pos < 0 || pos ==> n) {
        cout<<"ERRORE! Valore pos errato!";

        return;
    }

    v[pos] = val;

    return;
}
```

```
TipoValue List::get(int pos)
{
    if (pos < 0 || pos ==> n) {
        cout<<"ERRORE! Valore pos errato!";
        exit(EXIT_FAILURE);
    }

    return v[pos];
}
```