Capitolo 11

Astrazione sui dati

Astrazione e modularità

- **Componente**
  - Unità di programma
    - funzione, struttura dati, modulo
- **Interfaccia**
  - Tipi e operazioni definiti in un componente che sono visibili *fuori* del componente stesso
- **Specifica**
  - Funzionamento “inteso” del componente, espresso mediante proprietà osservabili attraverso l’interfaccia
- **Implementazione**
  - Strutture dati e funzioni definiti *dentro* al componente, non necessariamente visibili da fuori

Esempio: Tipo di dato

- Componente
  - Coda a priorità: struttura dati che restituisce elementi in ordine decrescente di priorità
- Interfaccia
  - Tipo **`PrioQueue`**
  - Operazioni **`empty`** : **`PrioQueue`**
    `insert` : **`ElemType`** * **`PrioQueue`** → **`PrioQueue`**
    `deletemax` : **`PrioQueue`** → **`ElemType`** * **`PrioQueue`**
- Specifica
  - **`insert`** aggiunge all'insieme di elementi memorizzati
  - **`deletemax`** restituisce l'elemento a max priorità e la coda degli elementi rimanenti

Astrazione sui dati

- Tipo di dato = valori e operazioni

  `integer = [-maxint..maxint]` e `{+,-,*,div,mod}`

  - le operazioni sono il solo modo di manipolare un integer
  - p.e. non sono possibili shift su valori integer

- Astrazione sui dati:

  La rappresentazione (implementazione)
  dei dati
  delle operazioni
  inaccessibile all'utente, perché protetta da una capsula che la isola

Quale supporto linguistico per l'astrazione ?

- Astrazione sul controllo
  - Nascondi la realizzazione nel corpo di procedure
- Astrazione sui dati
  - Nascondi decisioni sulla rappresentazione delle strutture dati e sull'implementazione delle operazioni
  - Esempio: una coda a priorità realizzata mediante
    - un albero binario di ricerca
    - un vettore parzialmente ordinato

- Quale supporto linguistico è fornito da un linguaggio a questo "nascondimento" dell'informazione (*information hiding*) ?

Tipi di dato astratti (Abstract Data Types)

- Uno dei maggiori contributi ai linguaggi degli anni ‘70
- Idea di fondo:
  - Separa l’interfaccia dall’implementazione
    - Esempio:
      - **`Sets`** hanno **`empty`**, **`insert`**, **`union`**, **`is_member?`**, …
      - **`Sets`** implementato come … vettore, lista concatenata …
  - Usa il controllo di tipo per garantire la separazione
    - Il cliente ha accesso alle sole operazioni dell’interfaccia
    - L’implementazione è *incapsulata* in opportuni costrutti *opachi* (ADT)

ADT per uno stack di interi

```
abstype Int_Stack{
   type Int_Stack = struct{
                       int P[100];
                       int n;
                       int top;
   }
   signature
      Int_Stack crea_pila();
      Int_Stack push(Int_Stack s, int k);
      int top(Int_Stack s);
      Int_Stack pop(Int_Stack s);
      bool empty(Int_Stack s);
   operations
      Int_Stack crea_pila(){
         Int_Stack s = new Int_Stack();
         s.n = 0;
         s.top = 0;
         return s;
      }
      Int_Stack push(Int_Stack s, int k){
         if (s.n == 100) errore;
         s.n = s.n + 1;
         s.P[s.top] = k;
         s.top = s.top + 1;
         return s;
      }
      int top(Int_Stack s){
         return s.P[s.top];
      }
      Int_Stack pop(Int_Stack s){
         s.n = s.n - 1;
         s.top = s.top - 1;
         return s;
      }
      bool empty(Int_Stack s){
         return (s.n == 0);
      }
}
```

Un'altra definizione per lo stack di interi


```
abstype Int_Stack{
   type Int_Stack = struct{
                       int info;
                       Int_stack next;
   }
   signature
      Int_Stack crea_pila();
      Int_Stack push(Int_Stack s, int k);
      int top(Int_Stack s);
      Int_Stack pop(Int_Stack s);
      bool empty(Int_Stack s);
   operations
      Int_Stack crea_pila(){
         return null;
      }
      Int_Stack push(Int_Stack s, int k){
         Int_Stack tmp = new Int_Stack(); // nuovo elemento
         tmp.info = k;
         tmp.next = s;                    // concatenalo
         return tmp;
      }
      int top(Int_Stack s){
         return s.info;
      }
      Int_Stack pop(Int_Stack s){
         return s.next;
      }
      bool empty(Int_Stack s){
         return (s == null);
      }
}
```

Principio di incapsulamento

- Indipendenza dalla rappresentazione

Due implementazioni corrette di un tipo (astratto) non sono distinguibili dai clienti di quel tipo

  - Le implementazioni sono modificabili senza con ciò interferire con alcun cliente

  - Perché il cliente non ha alcun modo per accedere all'implementazione

Moduli

- Costrutto generale per lo *information hiding*
  - disponibile in linguaggi imperativi, funzionali, ecc.
- Due parti
  - Interfaccia:

    Un insieme di nomi e relativi tipi
  - Implementazione:

    Dichiarazioni (di tipi e funzioni) per ogni nome dell'interfaccia

    Dichiarazioni aggiuntive nascoste (al cliente)
- Esempi:
  - moduli di Modula, packages di Ada, strutture di ML, ...