Programmazione e Strutture Dati (PR&SD)

I° ANNO – Informatica

Prof. V. Fuccella

Abstract Data Types (ADT)

Astrazione

- Procedimento mentale che consente di:
  1. Evidenziare le caratteristiche pregnanti di un problema
  2. Offuscare o ignorare gli aspetti che si ritengono secondari rispetto ad un determinato obiettivo

Tipi di astrazione
Funzionale e Procedurale

- **Finalità**: ampliare l'insieme dei modi di operare sui tipi di dati già disponibili, attraverso la definizione di nuovi operatori
- Una funzionalità di un programma è delegata ad un sottoprogramma (funzione o procedura)
  - È definita ed usabile indipendentemente dall'algoritmo che la implementa (es. algoritmi di ordinamento di un array)

Tipi di astrazione
Dati

- Ricalca ed estende il concetto di astrazione funzionale.
- **Finalità**: ampliare i tipi di dati disponibili attraverso l'introduzione sia di nuovi tipi di dati che di nuovi operatori.
  - definiti a prescinde dalla loro implementazione in uno specifico linguaggio di programmazione.

5

Tipi di Dati

- Un **tipo** di dati è definito da un **dominio di valori** e un insieme di **operazioni** previste su quei valori

  esempio: il tipo interi può essere definito da
  - un intervallo di valori interi $[-2^{m-1}, 2^{m-1} - 1]$
  - Le operazioni aritmetiche/logiche elementari {+, -, *, /, %, ==, !=, >, <, <=, >=}
- Nel linguaggio c
  - Tipi di dati **primitivi**: forniti direttamente dal linguaggio: int, char, float, double
  - Dati **aggregati**: array, strutture, enumerazioni, unioni
  - Puntatori

6

Tipi di Dati Astratti (ADT)

- Tipo di dati che estende dati esistenti, definito distinguendo **Specifica** e **implementazione**
- **Specifica**:
  1. Definizione del tipo di dati
  2. Definizione dell'insieme degli operatori
  - Sintattica: regole (nomi e tipi)
  - Semantica: significati (valori, vincoli)
- **Implementazione:**
  - Codifica di quanto definito nella specifica, usando primitive e costrutti di un linguaggio di programmazione
  - è spesso nascosta al programmatore, seguendo il principio dell'**Incapsulamento** (information hiding)

ADT
Specifica

	Sintattica	Semantica
Tipi di dati	• Nome dell'ADT • Tipi da dati già usati	• Insieme dei valori
Operatori: Per ogni operatore	• Nome dell'operatore • Tipi di dati di input e di output	Funzione associata all'operatore • Precondizioni: definiscono quando l'oeratore è applicabile • Postcondizioni: definiscono relazioni tra dati di input e output

ADT: Punto

Sintattica	Semantica
Nome del tipo: Punto Tipi usati: Reale	Dominio: Insieme delle coppie (ascissa, ordinata) dove ascissa e ordinata sono numeri reali
creaPunto (reale, reale) → punto	creaPunto(x, y) = p • pre: true • post: p = (x, y)
ascissa (punto) → reale	ascissa(p) = x • pre: true • post: p = (x, y)
ordinata (punto) → reale	ordinata(p) = y • pre: true • post: p = (x, y)
distanza (punto, punto) → reale	distanza(p1, p2) = d • pre: true • post: d = sqrt((ascissa(p1)-ascissa(p2))^2 + (ordinata(p1)-ordinata(p2))^2)

Implementazione ADT in C
Strutture

- **Definizione**: tipo di dati composito che include un elenco di variabili fisicamente raggruppate in un unico blocco di memoria
- Vantaggio: migliore leggibilità dei programmi

```
struct point {
   float x;
   float y;
};
```

```
int main(){
  struct point p;
  p.x = 2.0;
  p.y = 3.0;
  printf("coordinate del punto: (%.1f, %.1f)",
p.x, p.y);
}
```

Implementazione ADT in C
Strutture – Inizializzazione

- È possibile inizializzare una struttura…

```
struct point {
   float x;
   float y;
};

int main(){
  struct point p = {2.0, 3.0};
  printf("coordinate del punto: (%.1f, %.1f)", p.x, p.y);
}
```

Implementazione ADT in C
Typedef

- Si può usare il typedef sulla struttura precedentemente definita

```
typedef struct point Point;
```

- Oppure usare in combinazione il typedef e la definizione della struttura

```
typedef struct{
        float x;
        float y;
} Point;
```

```
int main(){
  Point p = {2.0, 3.0};
  printf("coordinate del punto: (%.1f, %.1f)",
p.x, p.y);
}
```

Implementazione ADT in C
Puntatori

- È possibile allocare memoria per una struttura attraverso le funzioni *malloc* o *calloc*

```
Point *p = malloc(sizeof(Point));
```

- È possibile accedere ai campi della struttura da un puntatore usando l'operatore freccia ->

```
int main(){
        Point *p = malloc(sizeof(Point));
        p->x = 2.0;
        p->y = 3.0;
        printf("coordinate del punto: (%.1f, %.1f)", p->x, p->y);
}
```

Implementazione
punto.h

```
typedef struct {
    float x;
    float y;
}  Punto;

punto creaPunto (float x, float y);
float ascissa (Punto p);
float ordinata (Punto p);
float distanza (Punto p1, Punto p2);
```

ADT Punto
Problemi

- L'implementazione della struttura del tipo punto è nell'header file
- Visibile quindi al modulo client, che potrebbe quindi accedere direttamente ai campi della struct senza usare gli operatori dell'ADT
- Come facciamo a "nascondere" la rappresentazione della struttura del tipo di dato, evitando di inserirla nell'header file ?

Rivediamo *punto.h*

```
typedef struct punto *Punto;

Punto creaPunto (float x, float y);
float ascissa (Punto p);
float ordinata (Punto p);
float distanza (Punto p1, Punto p2);
```

- Il tipo Punto è implementato come un puntatore alla struttura
  - NB: il nome del tipo e l'interfaccia degli operatori non sono cambiati, per cui il programma client non necessita di modifiche
- L'implementazione della struct punto è definita nel file punto.c in modo da non renderla visibile al client tramite l'include dell'header file punto.h

Una nota …

- Nell'header file punto.h non possiamo non dichiarare

  typedef struct punto *Punto;

  perché all'atto della compilazione del modulo client, il compilatore non saprebbe quanta memoria allocare per una dichiarazione del tipo:

  Punto p;

- Invece, essendo il tipo punto un puntatore, il compilatore sa quanta memoria deve allocare per una variabile di quel tipo, indipendentemente dalla dimensione dell'elemento puntato