Abstract Data Types (ADT)

Tipi di dati astratti

Astrazione Dati e Funzionale

- L’astrazione dati ricalca ed estende il concetto di astrazione funzionale. Così come l'astrazione funzionale permette di ampliare l'insieme dei modi di operare sui dati, cioè gli operatori sui tipi di dati già disponibili, l’astrazione di dati permette di ampliare i tipi di dati disponibili attraverso l'introduzione sia di nuovi tipi di dati che di nuovi operatori.

- L’astrazione funzionale stimola gli sforzi per evidenziare operazioni ricorrenti o ben caratterizzate all’interno della soluzione di un problema.

- L’astrazione di dati sollecita ad individuare le organizzazioni dei dati più adatte alla soluzione del problema.

Astrazione dati :
Specifica e Realizzazione

- ***Specifica*:** descrivere un nuovo tipo di dati e gli operatori applicabili
  - ***La specifica descrive l'astrazione dati e il modo in cui può essere utilizzata attraverso i suoi operatori***
  - ***Specifica sintattica e semantica …***
- ***Realizzazione:*** come il nuovo dato e i nuovi operatori vengono ricondotti ai dati e agli operatori già disponibili
  - ***Utilizzo di meccanismi di programmazione modulare offerti dal linguaggio di programmazione utilizzato per mettere a disposizione l'astrazione attraverso un'interfaccia e nascondere i dettagli dell'implementazione***

Specifica sintattica e semantica

- Specifica ***sintattica***
  - i nomi del tipo di dati di riferimento e degli eventuali tipi di dati usati (già definiti)
  - i nomi delle operazioni del tipo di dati di riferimento
  - i tipi di dati di input e di output per ogni operatore

Specifica sintattica e semantica

- Specifica ***semantica***
  - L'insieme dei valori associati al tipo di dati di riferimento
  - La funzione associata ad ogni nome di operatore, specificata dalle seguenti condizioni:
    - ***i) precondizione***: definita sui valori dei dati di input definisce quando l'operatore è applicabile
    - ***ii) postcondizione***: definita sui valori dei dati di output e di input, stabilisce la relazione tra argomenti e risultato

Esempio: il tipo di dato astratto *libro*

Specifica dei tipi di dati

- Specifica ***sintattica***
  - Tipo di riferimento: libro
  - Tipi usati: stringa, intero
- Specifica ***semantica***
  - Il tipo libro è l'insieme delle quadruple (autore, titolo, editore, anno) dove autore, titolo e editore sono stringhe e anno è un intero

Esempio: il tipo di dato astratto *libro*

Specifica degli operatori

- Specifica ***sintattica***
  - creaLibro(stringa, stringa, stringa, intero) ✉ibro
  - autore (libro) ✉stringa
  - titolo (libro) ✉stringa
  - editore (libro) ✉stringa
  - anno (libro) ✉ntero

Esempio: il tipo di dato astratto *libro*

Specifica degli operatori

- Specifica ***semantica***
  - creaLibro(aut, tit, ed, an) = lb
    - Post: lb = (aut, tit, ed, an)
  - autore(lb) = aut
    - Post: lb = (aut, tit, ed, an)
  - titolo(lb) = tit
    - Post: lb = (aut, tit, ed, an)
  - editore(lb) = ed
    - Post: lb = (aut, tit, ed, an)
  - anno(lb) = an
    - Post: lb = (aut, tit, ed, an)

Nel nostro esempio non ci sono precondizioni (o meglio la precondizione è sempre verificata)

NB: precondizioni e postcondizioni sono espressioni logiche … L'operatore "=" che compare in queste condizioni NON è un assegnamento !

Esempio: il tipo di dato astratto *libro*

Una possibile implementazione: file libro.h

```
typedef struct lib {
        char autore[26];
        char titolo[53];
        char editore[26];
        int anno;
}  libro;

libro creaLibro (char *aut, char *tit, char *ed, int anno);
char *autore (libro l);
char *titolo (libro l);
char *editore (libro l);
int  anno (libro l);
```

Esempio: il tipo di dato astratto *libro*

Una possibile implementazione: file libro.c

```
#include “libro.h”

libro creaLibro (char *aut, char *tit, char *ed, int anno)
{
        libro l;
        strcpy(l.autore, aut);
        strcpy(l.titolo, tit);
        strcpy(l.editore, ed);
        l.anno = anno;
        return l;
}
```

Esempio: il tipo di dato astratto *libro*

```
char *autore (libro L)
{
        char *aut;
        aut = calloc (26, sizeof(char));
        strcpy(aut, L.autore);
        return aut;
}

char *titolo (libro L)
{
        char *tit;
        tit = calloc (53, sizeof(char));
        strcpy(tit, L.titolo);
        return tit;
}
```

Esempio: il tipo di dato astratto *libro*

file libro.c

```
char *editore (libro L)
{
        char *ed;
        ed = calloc (26, sizeof(char));
        strcpy(ed, L.editore);
        return ed;
}

int  anno (libro L)
{
        return L.anno;
}
```

Esempio: il tipo di dato astratto *punto*

Specifica dei tipi di dati

- Specifica ***sintattica***
  - Tipo di riferimento: punto
  - Tipi usati: reale
- Specifica ***semantica***
  - Il tipo punto è l'insieme delle coppie (ascissa, ordinata) dove ascissa e ordinata sono numeri reali

Esempio: il tipo di dato astratto *punto*

Specifica degli operatori

- Specifica ***sintattica***
  - creaPunto (reale, reale) ✉ punto
  - ascissa (punto) ✉ reale
  - ordinata (punto) ✉ reale
  - distanza (punto, punto) ✉ reale

Esempio: il tipo di dato astratto *punto*

Specifica degli operatori

- Specifica ***semantica***
  - creaPunto(x, y) = p
    - pre: true
    - post: p = (x, y)
  - ascissa(p) = x
    - pre: true
    - post: p = (x, y)

      /* il risultato x è il primo elemento della coppia p */
  - …

NB: precondizioni e postcondizioni sono espressioni logiche …

L'operatore "=" che compare in queste condizioni NON è un assegnamento !

Esempio: il tipo di dato astratto *punto*

Specifica degli operatori

- Specifica ***semantica***
  - ordinata(p) = y
    - pre: true
    - post: p = (x, y)
  - distanza(p1, p2) = d
    - pre: true
    - post: d = sqrt((ascissa(p1)-ascissa(p2))^2 +
      (ordinata(p1)-ordinata(p2))^2)

Esempio: il tipo di dato astratto *punto*

Una possibile implementazione: il file 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);
```

Esempio: il tipo di dato astratto *punto*

Una possibile implementazione: il file punto.c

```
#include <math.h>
#include “punto.h”

punto creaPunto(float x, float y) {
        punto p;
        p.x = x;
        p.y = y;
        return p;
}

float ascissa(punto p) {
        return p.x;
}
```

```
float ordinata(punto p) {
        return p.y;
}

float distanza(punto p1, punto p2) {
        float dx = p1.x – p2.x;
        float dy = p1.y – p2.y;
        float d = sqrt(dx*dx +
dy*dy);
        return d;
}
```

Usiamo l'ADT punto

- Realizzare un programma che prende in ingresso una sequenza di punti e un numero reale d e restituisce il numero di m coppie di punti che hanno distanza minore di d

Specifica del programma

- Specifica
  - Dati di ingresso
    - una sequenza sp di punti
    - un numero reale d
  - Dati di uscita
    - un intero m
  - Precondizione
    - sp non vuota
  - Postcondizione
    - m è il numero di coppie di punti p1 e p2 in sp tali che distanza(p1, p2) < d

Progettazione (versione senza file)

1. Input numero di punti n
2. Crea array a di punti di dimensione n
3. Input n punti e carica in array a
4. Input distanza d
5. Calcola il numero m di coppie in a con distanza minore di d
6. Output m

I passi 1, 2 4 e 6 sono direttamente implementabili con istruzioni del linguaggio

Realizziamo i sottoprogrammi per i passi 3 e 5
Fare analisi e progettazione come esercizio …

Il main

file: vicini.c

```
#include <stdio.h>
#include <stdlib.h>
#include “punto.h”

void input_punti(punto a[], int n);
int coppie_vicine(punto a[], int n, float d);

int main()
{
        float d;
        int n, m;
        punto *a;                                   // array di punti

        printf(“Numero punti da caricare :”);
        scanf(“%d”, &n);                                    // passo 1
        a = malloc(n*sizeof(punto)) ;                       // passo 2
        input_punti(a, n);                                  // passo 3
        printf(“Inserisci distanza: ”);
        scanf(“%f”, &d);                                    // passo 4
        m = coppie_vicine(a, n, d);                         // passo 5
        printf(“Numero di coppie di punti con distanza”);
        printf(“ minore di %f: %d \n”, d, m);               // passo 6
    return 0;
}
```

input_punti

file: vicini.c

```
void input_punti(punto a[], int n)
{
        int i;
        float x, y;
    for(i = 0; i < n; i++)
        {
        printf( “Ascissa punto %d: ”, i);
        scanf(“%f”, &x);
            printf( “Ordinata punto %d:”, i);
        scanf(“%f”, &y);
            a[i] = creaPunto(x, y);
        }
}
```

coppie_vicine

file: vicini.c

```
int coppie_vicine(punto a[], int n, float d)
{
        int i, j, m;

        m = 0;
        for(i=0; i < n-1; i++)
            for(j=i+1; j<n; j++)
                if(distanza(a[i], a[j])<d)
                    m++;
        return m;
}
```

Progettazione (con uso di file)

1. Input nome del file sp contenente la sequenza di punti e distanza d // argomenti su riga di comando
2. Calcola numero di coppie n in sp
3. Carica i punti da file sp e inseriscili in array a di dimensione n
4. Calcola il numero m di coppie in a con distanza minore di d // come prima
5. Output m

Realizziamo i sottoprogrammi per i passi 2 e 3 …
Fare analisi e progettazione come esercizio …

Il main

```
#include <stdio.h>
#include <stdlib.h>
#include "punto.h"

int numero_punti(char *file_name);
void carica_punti(char *file_name, punto a[], int n);
int coppie_vicine(punto a[], int n, float d);


int main(int argc, char *argv[])
{
        float d;
        int n, m;
        punto *a;                                   // array di punti

        if (argc < 3) {
            printf("Numero parametri non corretto \n");
            exit(1);
            }
//  CONTINUA
```

Il main

```
        n = numero_punti(argv[1]);                    // passo 2

        if(n ==0)
        {
              printf("Nessun punto caricato dal file %s \n", argv[1]);
              exit(1);
        }

        a = malloc(n*sizeof(punto)) ;
        carica_punti(argv[1], a, n);                  // passo 3

        d = atof(argv[2]);

        m = coppie_vicine(a, n, d);                   // passo 4

        printf("Numero di coppie di punti nel file %s", argv[1]);
        printf(" con distanza minore di %f: %d \n", d, m);
    return 0;
}
```

numero_punti

```
int numero_punti(char *file_name)
{
        int n =0;
        FILE *sp;
        float x, y;

        sp=fopen(file_name, "r");

        if(sp!=NULL)
        {
            while(fscanf(sp, "%f%f" , &x, &y) == 2)
                n++;
            fclose(sp);
        }
        return n;
}
```

carica_punti

```
void carica_punti(char *file_name, punto a[], int n)
{
        int i= 0;
        FILE *sp;
        float x, y;

        sp=fopen(file_name, "r");

        if(sp==NULL)
            printf("Errore in apertura del file %s \n", file_name);
        else
        {
            while (fscanf(sp, "%f%f", &x, &y) == 2)
            {
                a[i] = creaPunto(x, y);
                i++;
            }
            fclose(sp);
        }
}
```

ADT Punto : problemi

- L'implementazione della struttura del tipo punto è nell'header file
  - Stessa cosa anche per l'ADT Libro
- 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 pto *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
- L'implementazione della struct pto è definita nel file punto.c in modo da non renderla visibile al client tramite l'include dell'header file punto.h

Rivediamo l'implementazione *nel file punto.c*

```
#include <math.h>
#include <stdlib.h>
#include "punto.h"

struct pto {
        float x;
        float y; };

punto creaPunto(float x, float y) {
        punto p =
malloc(sizeof(*p));
        p->x = x;
        p->y = y;
        return p;
}
```

```
float ascissa(punto p) {
        return p->x;
}

float ordinata(punto p) {
        return p->y;
}

float distanza(punto p1, punto p2) {
        float dx = p1->x – p2->x;
        float dy = p1->y – p2->y;
        float d = sqrt(dx*dx +
dy*dy);
        return d;
}
```

Note …

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

  ```
  typedef struct pto *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
- **NB: il nome del tipo e l'interfaccia degli operatori non sono cambiati, per cui il programma client non necessita di modifiche**

Esercizio: ADT Vettore

- Operatori
  - CreaVettore
    - Crea un vettore di n elementi
  - LeggiVettore
    - Legge l'elemento di posizione i del vettore
  - ScriviVettore
    - Modifica l'elemento in posizione i del vettore

Fare specifica e implementazione con array come esercizio ...