Tipi di dati strutturati
e
Linguaggio C

Record o strutture

Il costruttore struct in C

Dati strutturati

Record

Un ***record*** o ***struttura*** è una struttura dati ottenuta aggregando elementi di tipo diverso che potrebbero essere anch'essi strutturati

Esempi sono il ***numero complesso*** ottenuto da una coppia di reali o un ***punto*** ottenuto da una terna di numeri reali

Dichiarazione di una struttura in C

Dichiarazione di una struttura

typedef struct {

t1 s1;

t2 s2;

……..

tn sn;

} nome-tipo;

Un record è definito da un elenco di campi preceduto dalla parola riservata ***struct*** racchiusi tra parentesi graffe

Dichiarazione di una struttura in C

Esempi di dichiarazione di strutture:

```
typedef struct
{   float x ;
   float y;
   float z;
} punto;
punto p;
```

Dichiarazione di una struttura in C

Esempi di dichiarazione di strutture:

```
typedef struct
{ int giorno;
int mese;
int anno;
} data ;
data data_esame;
```

```
struct
{ int giorno;
int mese;
int anno;
} data_esame;
```

Struttura e Record

Il ***record o struttura*** è una struttura di dati non omogenei: è costituita da componenti di tipo differente

E' una struttura ad accesso casuale ***(random)***

– è possibile cioè selezionare qualsiasi componente

*Ogni componente di un record si chiama **campo**: un campo deve avere un nome e deve essere associato a un tipo*

Struttura: Selezione di una componente

Sia x una variabile di tipo struttura, avente campi *s1*.. *sn*

- ***Selezione (lettura)***

La *selezione* di una componente è possibile mediante l'identificatore della variabile ed il nome del campo separati dal ***simbolo .*** *(dot notation)*

x.s1 seleziona il campo s1 della variabile x
……..
x.sn seleziona il campo sn della variabile x

Struttura: Aggiornamento di una componente

Sia x una variabile di tipo struttura

- ***Aggiornamento (scrittura)***

L'aggiornamento della variabile x si ha, memorizzando un valore in una componente

x.s1 = v aggiorna la variabile x,

modificando il valore del campo s1

Esempi di Dichiarazioni

```
typedef struct
{
string nominativo;
string matricola;
data data_nascita
} studente;
```

```
typedef char string [30];
typedef struct
{ int giorno;
int mese;
int anno;
} data;
```

Esempi di Selezione di Campi

studente dati [100];

dati *è il nome di un vettore di 100 componenti di tipo* ***studente***

dati [i] *per i che varia da 0 a 99 seleziona la componente i del vettore*

dati[i].matricola *seleziona il campo matricola della componente dati[i]*

dati[i].data_nascita.anno *seleziona il campo anno del campo*

data_nascita della componente dati[i]

Assegnamento

Siano a e b due strutture dello steso tipo

E’ possibile scrivere una istruzione di assegnamento globale tra le 2 strutture a, b

```
data data_esame, data_odierna;
……..
data_esame = data_odierna;
```

Approfondimenti Linguaggio C

TIPI DI DATI

Il linguaggio C prevede tipi di dati ***non strutturati*** e ***strutturati***

- **Tipi di dati *non strutturati***

I 4 *tipi base* sono **char**, **int**, **float**, **double**

Possono essere modificati mediante 4 ***qualificatori***:

- ***short***, per dati di tipo **int**, forza la rappresentazione degli interi su 16 bit
- ***long***, per dati di tipo **int** e **double**, forza la rappresentazione su 32 e 80 bit rispettivamente
- ***signed*** e ***unsigned***, per dati di tipo **int** e **char**, specificano dati con segno o senza segno

TIPI DI DATI

Il linguaggio C prevede tipi di dati ***non strutturati*** e ***strutturati***

- **Tipi di dati *strutturati***

Gli array e le stringhe sono esempi di tipi di dati strutturati.
Altri esempi sono il tipo "***enum***" e il tipo "***struct***"

Problema: simulare il tipo di dati logico (booleano)

Soluzione 1

```
#define TRUE   1
#define FALSE  0
int espressione;
…
if (espressione = =
TRUE)
…
```

Soluzione 2

```
enum logical {FALSE, TRUE};
enum logical trovato;
…
if (trovato = = TRUE)
…
```

Soluzione 3

```
typedef enum {FALSE, TRUE} boolean;
boolean trovato;
…
if (trovato = = TRUE)
…
```

IL TIPO DI DATI ENUM

a) `enum nome_tipo {lista identificatori};`

b) `enum nome_tipo variabile;`

a) identifica con `nome_tipo` la lista dei valori che possono essere assunti

b) definisce `variabile` di tipo enumerativo `nome_tipo`;
`variabile` potrà assumere solo i valori prima specificati

TYPEDEF

Sintassi:

```
typedef   tipo_esistente   nome_nuovo_tipo;
```

`typedef` consente di ridefinire il nome di un tipo di dati esistente

STRUTTURE

Esempio:

Un programma deve manipolare date nella forma

31 maggio 2000

una possibile organizzazione dei dati è:

giorno	**31**
mese	**maggio**
anno	**2000**

STRUTTURE

Tale tipo di organizzazione può essere ottenuta con una **STRUTTURA**, ossia un ***insieme di più variabili***, ***anche di tipo diverso***,
logicamente connesse sotto un singolo nome

(in Pascal si chiamano record)

- servono per trattare **dati complessi**
- permettono di trattare come **unica entità** un insieme di **variabili correlate**

DEFINIZIONE DI UNA STRUTTURA

Le strutture in C sono definite mediante l'istruzione **struct**

Esempio

```
struct data {
int Giorno;
char Mese[9];
int Anno;};
```

La definizione di una struttura
non crea una variabile, ma **un tipo di dato**
per future dichiarazioni di variabile.

Nell'esempio, l'***identificatore della struttura*** **data** è il nome che verrà usato per la dichiarazione di nuove istanze di variabili di tipo **struct data**

DICHIARAZIONE DI UNA VARIABILE DI TIPO STRUTTURA

1)

```
struct data {       /* definizione della struttura data */
 int Giorno;
 char Mese[9];
 int Anno;};

struct data Oggi;  /* dichiarazione della variabile di
                      tipo struct data */
```

2)

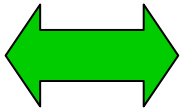
```
struct data {       /* definizione della struttura data */
 int Giorno;
 char Mese[9];
 int Anno;} Oggi;  /* e dichiarazione della variabile di
                      tipo struct data */
```

DICHIARAZIONE DI UNA VARIABILE DI TIPO STRUTTURA

3)

```
struct {                  /* definizione di una struttura */
   int Giorno;
   char Mese[9];
   int Anno;} Oggi;       /* e dichiarazione della variabile di
                             tipo struct data */
```

Sintatticamente:

`struct {……} x;`  **`int x;`**

SELEZIONE DEI MEMBRI DI UNA VARIABILE DI TIPO STRUTTURA

nomestruttura . nomemembro

Ad esempio:

Se **Oggi** è una variabile di tipo **struct data**:

Oggi.Giorno indica il membro Giorno;

Oggi.Mese indica il membro Mese;

Oggi.Anno indica il membro Anno.

Gli elementi di una struttura si chiamano ***membri***

ESEMPIO: conteggio degli studenti bocciati

```
#define DIM_CLASSE 100
struct studente
{ char *cognome;
  int  matricola;
  int  voto;};
int bocciati(struct studente classe[]);
void main()
{
   struct studente classe[DIM_CLASSE];
   …
}
int bocciati(struct studente classe[])
{
  int i, cnt=0;
  for (i=0; i<DIM_CLASSE;i++)
    if (classe[i].voto < 18)
      cnt++;
  return cnt;
}
```

INIZIALIZZAZIONE DI UNA VARIABILE DI TIPO STRUTTURA

Le strutture possono essere inizializzate *al momento della loro dichiarazione*:

Esempio

```
struct data Oggi={31, "maggio",2000};
```

oppure:

Esempio (2)

```
struct data Oggi;
Oggi.Giorno = 31;
Oggi.Mese = “maggio”;
Oggi.Anno = 2000;
```

Il membro di una struttura può essere a sua volta una struttura, fino a costruire strutture di strutture molto complesse

Esempio

```
struct persona{
      char NomePers[30];
      struct {
            char Via[30];
            int NumCiv;
            int CAP;
            char Citta[30];} Indirizzo;
      double Salario;
      struct data NascPers;
      struct data AsszPers;};
```

Se si dichiara **Impiegato** come variabile di tipo **struct persona**:

Impiegato.NascPers.Mese

è il mese di nascita dell'Impiegato

PUNTATORI A STRUTTURE

Ogni operazione possibile sulle strutture può essere eseguita utilizzando i ***puntatori a strutture***

Esempi

```
struct data *pdata;   /*dichiarazione di puntatore a struttura*/
struct data Oggi;     /*dichiarazione di istanza di struttura*/

*pdata = Oggi;        /*assegna alla struttura puntata da pdata
                        la struttura Oggi */

pdata = &Oggi;        /*assegna al puntatore l'indirizzo di
                        Oggi*/
i=(*pdata).Giorno;    /*Accesso ai membri di Oggi*/
```

Per accedere ai **membri** di una struttura mediante puntatori generalmente si usa la notazione:

```
pdata ->Giorno   ⟺   (*pdata).Giorno
```

L'operatore -> è associativo da sinistra a destra:

```
p ->q->m   ⟺   (p->q)->m
```

Gli operatori (), [], -> sono quelli con precedenza maggiore di tutti gli altri

```
++p ->x   ⟺   ++(p->x)
```

equivale ad **incrementare di 1** il valore del membro **x** della struttura puntata da **p**

Le strutture possono essere passate come argomento a funzioni

Esempio

```
…
struct data {……} Oggi;
…
in=funz(Oggi, n, …);
…
```

Come per tutti gli altri parametri,
il passaggio di una struttura avviene **per valore**

Affinché una funzione possa modificare i valori di una struttura occorre passare **l'indirizzo** della struttura

```
in=funz(&Oggi, n, …);
```

Le strutture possono essere utilizzate per i valori di ritorno di funzioni

Esempio

```
#include <stdio.h>
struct punto costruisci(float x, float y);
main(){
struct punto {          /* definizione della struttura */
       float x;
       float y;} p1;    /* dichiarazione di struttura */
float a,b;
scanf("%f %f",&a,&b);
p1=costruisci(a,b);
printf("p1=(%f,%f)\n",p1.x,p1.y);
}                       /* fine main */
struct punto costruisci(float x, float y) {
struct punto temp;
temp.x=x;
temp.y=y;
return temp;
}
```

ESEMPIO: USO DELLE STRUTTURE

Considerata la seguente struttura:

```
struct studente{
        char Nome[30];
        int NumMatr;
        int NumEsa;};
```

scrivere un programma che crei un elenco contenente i dati di *n* studenti

elenco => array1D

n studenti => allocazione dinamica !!

SOLUZIONE

```
#include <stdio.h>
#define MAX_NOME 30
struct studente{
   char Nome[MAX_NOME];
   int NumMatr;
   int NumEsa;};
main()
{
int i,n;
struct studente *el;
scanf(“%d%,&n);
el=(struct studente *) malloc(n*sizeof(struct studente));
   for (i=0;i<n;i++)
     scanf("%s %d %d", &(el+i)->Nome,
                &(el+i)->NumMatr, &(el+i)->NumEsa);
   for (i=0;i<n;i++)
     printf("%s %d %d", (el+i)->Nome,
                (el+i)->NumMatr, (el+i)->NumEsa);
free(el);
}
```

Parametri della funzione main

Esempio:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  int i;
  printf("Num. di argomenti: %d\n", argc);
  printf("Gli argomenti sono:\n");
  for (i=0; i<argc; i++)
    printf("%s\n",argv[i]);
}
```

Se il programma eseguibile è denominato "prog", la linea di comando:

prog arg2 arg3 arg4 arg5

...

restituisce il seguente output:

```
Num. di argomenti: 5
Gli argomenti sono:
prog
arg2
arg3
arg4
arg5
```

I Parametri della funzione main( )

```
int main(int argc, char *argv[])
{
...
}
```

La funzione main( ) ha 2 argomenti di nome prefissato:

- **argc = intero che fornisce il numero di argomenti della linea di comando;**
- **argv = array di puntatori a carattere (visto come array di stringhe) ciascuna stringa contiene uno degli argomenti della linea di comando (argv[0] contiene il nome del comando stesso)**