Programmazione I

a.a 2008-2009

docente: Carmine Gravino

Strutture

Presentazione realizzata dal Prof. Andrea De Lucia

Agenda della lezione

- ✓ Argomenti in linea di comando
- ✓ Strutture
- ✓ Albero binario
- ✓ Typedef

Argomenti alla linea di comando

- Nel Linguagggio C anche il main rappresenta una funzione per cui è possibile passargli degli argomenti che corrisponderanno poi ai parametri passati in linea di comando.
- Esiste comunque una modalità ben precisa di passaggio dei parametri. In particolare la funzione main riceve due argomenti:
- main(int argc, char *argv[])
- Il primo contiene il numero degli argomenti passati in linea di comando, il secondo è un vettore di stringhe che contiene gli argomenti passati uno per stringa.

Argomenti alla linea di comando

- Per convenzione gli argomenti passati in linea di comando sono separati da spazi bianchi, inoltre il nome del programma stesso rappresenta in assoluto il primo parametro passato, per cui l'argomento argc vale sempre almeno 1 e l'elemento argv[0] contiene sempre il nome del programma invocato.
- Se quindi argc vale 1 il main non ha argomenti, altrimenti argv[1] contiene il primo argomento ed argv[argc-1] contiene l'ultimo argomento.
- Il C assume che il contenuto di argv[argc] sia un puntatore nullo.

Esempio

- Scriviamo un programma echo che stampa a video i parametri passati.

```
main(int argc, char *argv[])
{
  int i;
  for( i = 1 ; i < argc ; ++i)
     printf("%s%s", argv[i], (i < argc-1) ? " " : "");
  printf("\n);
  return 0;
}
```

Esercizio

1. Scrivere la funzione get_token che stampa di un certo insieme di linee in input quella/e in cui ricorre un token passato come argomento al main.

Strutture

- Esistono due grandi categorie di dati in C:

  Non strutturati

  - int, char, float, pointer

  Strutturati

  - Omogenei
    - array
  - Eterogenei
    - struct, union

- Una **struct** C è una collezione di una o più variabili normalmente di tipo diverso raggruppate sotto lo stesso nome.
- Per dichiarare una struttura si usa il prefisso **struct**

Perché usare le strutture

- Quando in un programma si vogliono rappresentare dati appartenenti al mondo reale, i tipi di dati quali gli interi, i caratteri, ecc. non sono sufficienti.
- Spesso ci si deve riferire ad entità che sono collezioni di oggetti diversi, come ad esempio un libro è una entità composta da un titolo, un autore, un editore, il testo.
- Per rappresentare e gestire queste collezioni di dati, le strutture sono l'oggetto ideale, perché danno la possibilità di incorporare le diverse informazioni elementari che caratterizzano la collezione di dati.

Esempio di struttura

```
/* Struttura di elementi logicamente correlati */
struct dipendente {
        int codice;
        char nome[30];
        float stipendio;
}
```

- La parola **dipendente** non è una variabile. E' solamente un identificatore della nuova struttura in maniera da identificarla univocamente.
- La definizione di una **struct** crea solo una guida alla memoria che una variabile di tipo **struct** occuperà
  - ☞codice | nome |stipendio
  - ☞0 3 4 33 34 37

Dichiarazione di variabili struct

- Dichiarazione di variabile dipendente:
  - ☞ struct dipendente prgrmt;
- "*programmatore*" è una variabile di tipo struct dipendente
- Le struct possono essere inizializzate

struct dipendente prgrmt = {487,"Marco",10.000}

- oppure:

```
struct dipendente {
    int codice;
    char nome[30];
    float stipendio;
} programmatore = { 487, "Marco", 10.000 }
```

Assegnamento di valori a strutture

- Per accedere al singolo elemento di una **struct** bisogna usare il nome della variabile e il nome del campo nella struttura in questo modo

<nome variabile>.<nome campo>

- Esempio:

```
programmatore.codice = 487;
strcpy(programmatore.nome ,"Marco");
programmatore.stipendio = 10000;
```

- L'operatore "." consente di connettere il nome della struttura con quella del membro

Strutture ed array

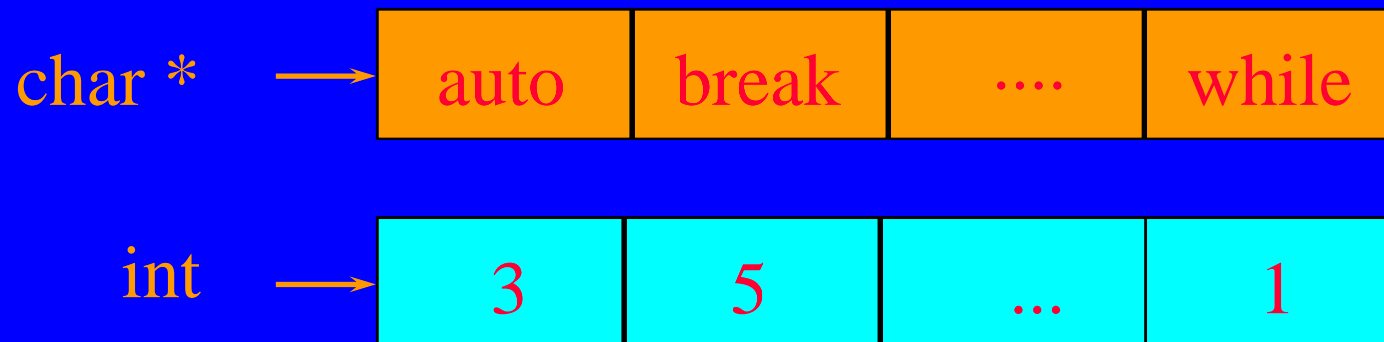
La dichiarazione di un array di tipo **struct** deve essere fatta come mostrato:

```
struct matita {
int durezza;
char *fabbricante;
int numero;
}
main()
{
   struct matita  m[3];
   m[0].durezza = 2;
   m[2].numero = 400;
   m[1].fabbricante = “Modini”;
}
```

```
struct matita {
int durezza;
char *fabbricante;
int numero;
} m[3];
```

Esempio

- Gli array di strutture sono utili al posto dei vettori paralleli o bidimensionali. Ad esempio un programma che conta le occorrenze delle parole chiavi del C potrebbe essere organizzato con due vettori paralleli:



- oppure attraverso un vettore di strutture in cui ogni elemento che descrive una parola chiave è rappresentato dai due membri della struct

Strutture nidificate

- Le strutture possono contenere all'interno anche altre strutture:

```
struct progetto {
    int mesi;
    double budget;
    char argomento[100];
    struct dipendente pgm[100];
} prg_soft;
```

- L'inserimento del 15° programmatore nel progetto può avvenire così:

```
strcpy(prg_soft.pgm[14].nome,"Marco");
```

Esempio

- Rappresentiamo attraverso una struttura l'oggetto rettangolo
- In una rappresentazione cartesiana un rettangolo è univocamente determinato dai suoi vertici in alto a destra ed in basso a sinistra. Per cui:

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

```
struct rectangle {
        struct point pt1;
        struct point pt2;
};
```

Puntatori a strutture

- Una struttura è un identificatore di tipo come qualsiasi altro, per cui è possibile anche definire dei puntatori a strutture. Ad esempio:

```
main()
{
struct matita {
        int durezza;
        char *fornitore;
} *mat_punt;     oppure   struct matita *mat_punt;
(*mat_punt).durezza = 3;
strcpy( (*mat_punt).fornitore, “Mondini”);
}
```

Puntatori a strutture

- Le parentesi nell'esempio precedente sono importanti in quanto la precedenza dell'operatore di accesso "." è superiore a quella di "*", per cui la dizione

*mat_punt.durezza equivale a *(mat_punt.durezza)

- che è sbagliato, per cui occorre scrivere:

(*mat_punt).durezza

Puntatori a strutture

- Per semplificare l'accesso ai membri di una struttura quando si usano i puntatori il C mette a disposizione l'operatore "->"

  <puntatore struttura> **->** *<membro struttura>*

- Per cui nell'esempio precedente potremo scrivere:
  - ☞mat_punt->durezza = 3:
  - ☞mat_punt->fornitore = "Mondini";
- Nel caso di strutture nidifcate l'operatore -> può essere utilizzato in ripetizione. Ad esempio

```
struct cartella { ....
   struct matita *m;   } *q;
q->m->fornitore        q->m->durezza
```

Structs come argomento di funzioni

- Una struct non è un array quindi viene passata alla funzioni per valore come qualsiasi altra variabile e non per riferimento.
- Ciò significa che se passiamo ad una funzione una struttura, tale funzione potrà operare solo su una copia per cui eventuali manipolazioni dei membri non avranno effetto sulla struttura stessa ma solo sulla sua copia.
- Se voglio passare a una funzione l'indirizzo di una **struct** basta specificare l'operatore **&** di fronte al nome:
  - ☞ assunzione (& programmatore);
- oppure passare un puntatore alla struttura …

Esercizio

1. Scrivere un insieme di funzioni di gestione dei rettangoli, in particolare una funzione che dati una coppia di punti crei un rettangolo, una funzione che riceve un rettangolo e ne calcola l'area, un'altra che ne calcola il perimetro, ed una che valuta se un punto passato come parametro cade all'interno del rettangolo oppure no.

Aritmetica dei puntatori a strutture

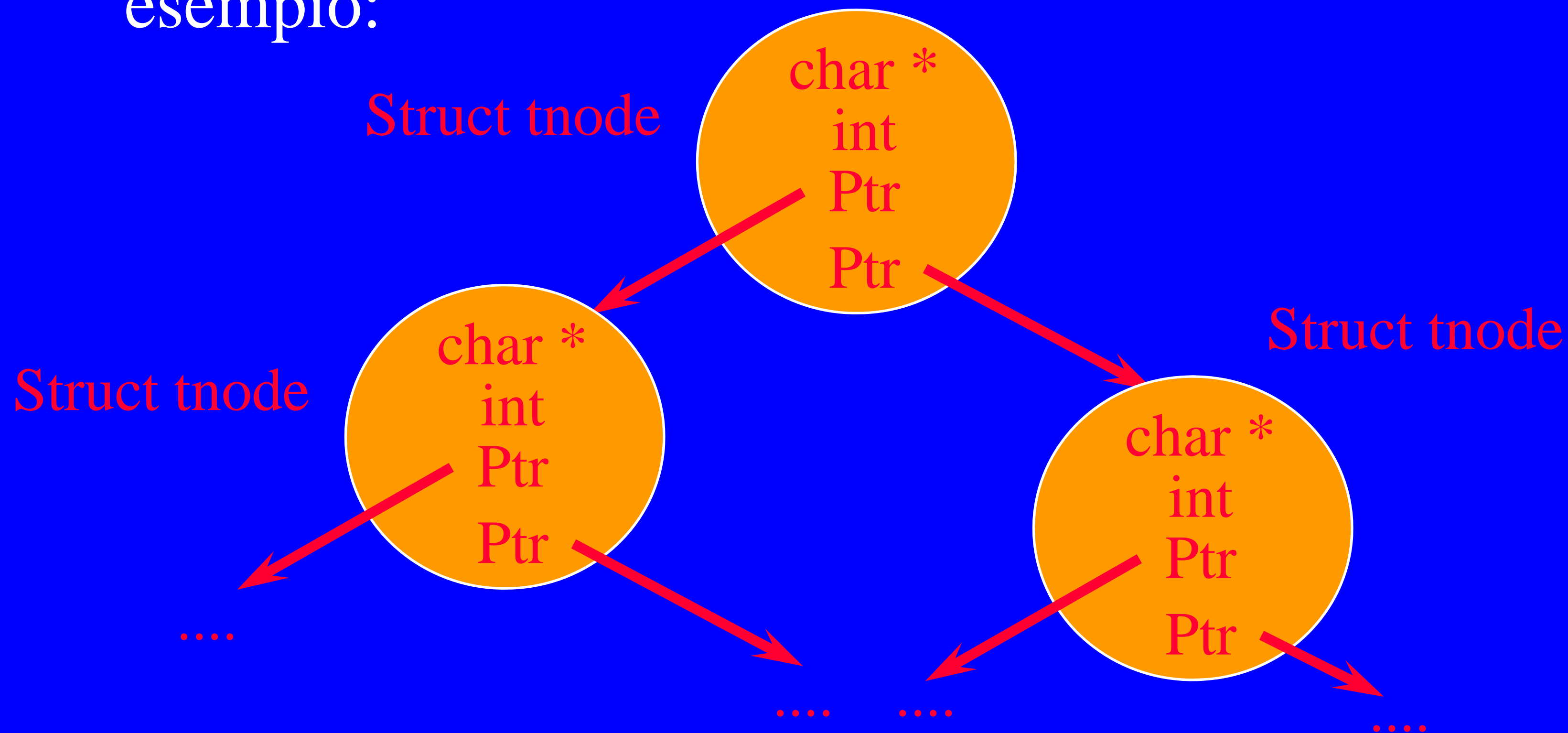
- Sui puntatori a strutture è possibile applicare un'aritmetica, così come fatto per gli altri puntatori. Occorre soltanto tener presente che gli operatori d'accesso "." e "->" hanno la precedenza massima e che sono operatori associativi da sinistra verso destra. Per cui:

```
struct {
        int len;
        char *str;
} *p;
```

++p->len **è diverso da**
++(p->len) **e da**
(++p)->len **e da**
++p(->len)   **SBAGLIATO**
*p->str++ **è diverso da**
(*p->str)++ **e da …**

Strutture ricorsive : Albero binario

- Un albero binario è una struttura ricorsiva che contiene oltre ai membri anche due puntatori (riferimenti) a strutture dello stesso tipo. Ad esempio:



Strutture ricorsive : Albero binario

- La cui implementazione è:

```
struct tnode {
        char *word;
        int count;
        struct tnode *left;
        struct tnode *right;
};
```

- ATTENZIONE : Notate che la struttura possiede una nidifcazione di se stesso, ma ciò si può ottenere soltanto utilizzando i riferimenti e non le strutture stesse. Infatti se utilizzassimo la struttura al posto del puntatore il compilatore non sarebbe in grado di risolvere l'ambiguità rappresentata dalla definizione di un tipo all'interno della sua dichiarazione.

Esempio

- Supponiamo di voler ordinare una lista di interi di lunghezza sconosciuta man mano che vengono inseriti in input.
- Supponiamo di utilizzare una struttura ad albero binario costruita man mano che vengono digitati i dati in input.
- La struttura albero binario possiede un dato intero e due puntatori, sinistro e destro, a strutture analoghe.
- All'atto dell'inserimento ciascun numero viene confrontato con quello presente nel nodo radice. Se risulta minore o uguale, lo si pone nel sottoalbero di sinistra, ovvero si crea una struttura il cui riferimento è il puntatore di sinistra della struttura radice; se maggiore lo si pone nel sottoalbero di destra.

Albero binario : implementazione

- Le due funzioni che gestiscano la struttura albero binario sono:
  - ☞struct tnode *addtree(struct tnode *,int);
  - ☞void treeprint(struct tnode *);
- La prima aggiunge elementi alla struttura secondo il criterio prima enunciato, la seconda stampa l'albero seguendo sempre prima la direzione dei sottoalberi destri.

```
struct tnode {
        int x;
        struct tnode *left;
        struct tnode *right;
};
```

Albero binario : funzione addtree

```
struct tnode *addtree(struct tnode *p,int x)
{
  if( p == NULL)              /* crea un nuovo nodo */
  {
        p = (struct tnode *)malloc(sizeof(struct tnode));
        p->x = x;
        p->left = p->right = NULL;
  }
  else if(x <= (p->x))
        p->left = addtree(p->left,x);
  else
        p->right = addtree(p->right,x);
  return p;
}
```

Albero binario : funzione treeprint

```
void treeprint(struct tnode *p)
{
  if( p != NULL)
  {
        treeprint(p->left);
        printf("%d\t",p->x);
        treeprint(p->right);
  }
}
```

Esercizio

1. Modificare la struttura albero binario precedente in modo da poter utilizzare stringhe di caratteri in luogo degli interi. Prevedere inoltre la gestione delle parole ripetute attraverso un contatore.

Typedef

- La dichiarazione **typedef** consente di dichiarare nuovi tipi sulla base di tipi esistenti.

typedef char *Stringa

- definisce un tipo puntatore a carattere che definiamo stringa, in questo modo sarà possibile:

Stringa str1,str2;

- Attenzione con typedef non si dichiara nessun oggetto, ma semplicemente si definiscono dei particolari identificatori per particolari tipi di dati. In pratica typedef si comporta analogamente ad una classe di memoria.

Typedef

- L'uso del typedef consente da un lato di aumentare la chiarezza e la leggibilità del programma, dall'altro consente di elevare la portabilità del programma stesso, in quanto per i tipi dipendenti dall'architettura della macchina, spostando il codice sarà sufficiente modificare le typedef senza entrare nel codice per modificare le dichiarazioni di variabili.

```
typedef struct punto {......} PIXEL;
```

- E' possibile ora dichiarare variabili del nuovo tipo definito con **typedef**:

```
PIXEL center , *origin;
```

- invece di:

```
struct punto center, *origin;
```