Vettori stringa, Dati strutturati, Funzioni, Puntatori

Vitoantonio Bevilacqua

bevilacqua@poliba.it

Parole chiave: Vettore stringa, Dichiarazione struttura dati, Definizione funzione, Puntatore a variabile, Puntatore a vettore.

1 Stringhe

Il concetto di "vettore" si può applicare anche a tipi di dati diversi da `int`: infatti così come una variabile di tipo `char` consente di memorizzare un singolo carattere, una "stringa", ossia un vettore di caratteri, consente di memorizzare una sequenza di caratteri trattandoli come una sola unità . L'espressione utilizzata per dichiarare una stringa di 10 caratteri è la seguente:

```
char frase[10];
```

Le differenze principali tra un vettore di interi e un vettore di caratteri sono che quest'ultimo è pensato per contenere testo e che automaticamente in una stringa vengono memorizzati i caratteri inseriti più uno, il carattere "\0" detto "carattere di fine stringa" che chiude la stringa. Dunque in realtà in `frase[10]` è possibile memorizzare massimo 9 caratteri poiché l'ultimo deve essere riservato per il carattere di fine stringa.

1.1 Funzioni printf e scanf con le stringhe

La presenza in ogni stringa del carattere di fine stringa, posto immediatamente dopo l'ultimo carattere memorizzato, permette di utilizzare funzioni come `scanf` e `printf` in modo ottimizzato.

Per poter visualizzare un'intera stringa è possibile usare l'istruzione printf tramite la specifica del formato %s nel seguente modo:

```
printf("%s",frase);
```

in questo caso l'istruzione `printf` stessa si comporta come se fosse un ciclo che stampa tutti gli elementi del vettore uno dopo l'altro fino al momento in cui non si raggiunge il carattere \0.

Analogamente per memorizzare una sequenza di caratteri in una stringa è possibile usare l'istruzione scanf nel seguente modo:

```
scanf("%s",frase);
```

così come per la `printf`, la `scanf` si comporta come un ciclo di acquisizione che memorizza carattere per carattere (esclusi gli spazi) nel vettore stringa, purché la sua lunghezza sia maggiore del numero di caratteri da memorizzare.

Sebbene per visualizzare o memorizzare un singolo carattere è possibile usare rispettivamente le funzioni `printf` e `scanf` con lo specificatore di formato %c, per queste e altre operazioni su singoli caratteri è opportuno in C utilizzare apposite funzioni incluse nella libreria `conio.h` come la funzione `getch()`, da utilizzare per memorizzare un singolo carattere, e la funzione `putch()`, da utilizzare per visualizzare un singolo carattere; queste si utilizza nel seguente modo:

```
char lettera;
lettera = getch();
putch(lettera);
```

1.2 Funzioni da libreria

Esistono nella libreria `string.h` funzioni che permettono di effettuare vari tipi di operazioni sulle stringhe; le funzioni più usate incluse nella libreria `string.h` sono:

- la funzione `strcpy` consente di copiare stringa2 su stringa1:

```
strcpy(stringa1,stringa2);
```

- la funzione strncpy consente di copiare i primi n caratteri di stringa2 in stringa1:

```
strncpy(stringa1,stringa2,n);
```

- la funzione strcmp consente di confrontare stringa2 con stringa1 e restituisce risultato intero uguale a 0 se le due stringhe sono uguali, positivo se stringa1 è maggiore di stringa2, altrimenti negativo:

```
strcmp(stringa1,stringa2);
```

- la funzione strcat consente di concatenare stringa2 a stringa1:

```
strcat(stringa1,stringa2);
```

- la funzione strlen restituisce la lunghezza della stringa:

```
strlen(stringa);
```

2 Dati strutturati

Finora abbiamo sempre operato con tipi di dati fondamentali, int, char, float, double, tipi di dati "built in" ossia già presenti all'interno del compilatore. Tuttavia a partire dai tipi fondamentali è possibile costruirsi nuovi tipi di dati, detti dati derivati o "dati strutturati". Ciò rappresenta la potenzialità di progettarsi un tipo di dato specifico.

Un tipo di dato strutturato viene sempre dichiarato tra le inclusioni delle librerie e il main; per dichiarare un tipo di dato strutturato si usa la parola chiave `struct` seguita dal nome identificativo del tipo di dato e successivamente all'interno delle parentesi graffe (da notare il punto e virgola in coda alla definizione di una struttura, immediatamente dopo la parentesi graffa chiusa!) si definiscono le "variabili interne", ossia i campi che compongono il dato. Nell'esempio sottostante si definisce un tipo di dato strutturato denominato "studente" contenente tre stringhe corrispondenti al nome, cognome e matricola, e un intero in cui dovrà essere memorizzato il voto:

```
struct studente
{
        char cognome[20],  nome[20], matricola[10];
        int voto;
};
```

D'ora in poi sarà possibile, nel main, definire dati di tipo studente:

```
struct studente Alunno;
```

con quest'istruzione viene definita una variabile Alunno composta da 54 byte consecutivi in memoria (20 per "nome", 20 per "cognome", 10 per "matricola", 4 per "voto").

Per accedere al campo di una variabile di tipo struttura si fa uso dell'operatore "." posto tra il nome della variabile strutturata e il nome del campo al quale si vuole accedere, quindi lo si può manipolare come qualsiasi altra variabile di tipo fondamentale:

```
Alunno.voto = 25;
Alunno.cognome = "Rossi";
Alunno.nome = "Mario";
printf("%s%s:%d",Alunno.cognome,Alunno.nome,Alunno.voto);
```

quest'istruzione assegna all'intero "voto" di Alunno il valore numerico 25, alla stringa "cognome" la sequenza di caratteri "Rossi" e alla stringa "nome" la sequenza di caratteri "Mario", quindi stampa a video il voto preso dall'alunno Mario Rossi. Dunque la sintassi generale con cui si fa riferimento al campo di un dato di tipo struttura è: `nome_variabile_struttura.nome_membro`.

Talvolta ci si può riferire ad una variabile struttura nel suo insieme soltanto attraverso il suo nome nel caso in cui non si vuol far riferimento ai singoli membri; in C è infatti possibile fare un'assegnazione tra variabili di tipo struttura, purché siano dello stesso tipo:

```
struct studente Alunno1, Alunno2;
...
Alunno1 = Alunno2;
```

tale istruzione permette di copiare tutti i valori di tutti i campi di Alunno2 nei corrispondenti campi di Alunno1.

Così come per interi e caratteri, anche ai dati strutturati si può estendere il concetto di vettore:

```
struct studente classe[100];
```

quest'ultima istruzione definisce nel main un vettore "classe" di 100 elementi, ciascuno dei quali è un dato di tipo studente; analogamente a quanto detto prima, per accedere ai campi dell'i-esimo elemento di classe, si utilizza l'operatore ".":

```
scanf("%s",classe[i].matricola);
```

3 Funzioni

Finora in tutti i programmi che abbiamo scritto, ogni istruzione corrispondeva ad una azione elementare. Ora scopriremo che in C è possibile unire gruppi di istruzioni per

formare delle “funzioni”, usate per evitare di replicare porzioni di codice sorgente più volte.

Invocare una funzione significa mandare in esecuzione la porzione di codice che la costituisce. Se una funzione è invocata più volte, allora la porzione di codice è eseguita più volte: il vantaggio è dunque quello di poter avere tante chiamate ma una sola porzione di codice.

3.1 Dichiarazione di una funzione

In C per dichiarare una funzione si usa la seguente sintassi:

```
tipo_ritorno nome_funz(tipo_par1,...,tipo_parn);
```

La dichiarazione di una funzione è posta tra l’inclusione delle librerie o tra l’eventuale dichiarazione di un tipo di dato strutturato e il main, e serve a specificare il tipo di dato che la funzione dovrà restituire quando essa è chiamata (codominio), il nome della funzione e tra parentesi tonde il tipo di ogni singolo argomento in ingresso che la funzione deve accettare (dominio). Esempio:

```
int cubo(int);
```

quest’istruzione dichiara una funzione “cubo” che accetta un argomento di tipo intero e restituisce un valore di tipo intero.

3.2 Definizione di una funzione

Dopo il main, la funzione deve essere definita; in C per definire una funzione si usa la seguente sintassi:

```
tipo_ritorno nome_funz (tipo_par1 par1,…,tipo_parn parn)
{
          ...;
          return espressione;
}
```

La definizione di una funzione stabilisce il tipo di dato in uscita, il nome della funzione, i “parametri formali”, ossia le variabili locali alla funzione in cui vengono memorizzati i valori in ingresso all’atto della sua chiamata, il blocco di istruzioni che ne costituiscono il contenuto e l’eventuale valore di ritorno.

Nelle parentesi tonde che seguono il nome della funzione sono specificati il tipo e il nome dei parametri formali.

Nel blocco di istruzioni delimitato da parentesi graffe può essere inserita qualunque istruzione compresa una chiamata di funzione. Si noti che una variabile dichiarata all’interno di un blocco istruzioni di una funzione è “locale”, in altre parole ogni funzione, compresa il main (la “funzione principale”), agisce su un area di memoria diversa, quindi nel caso sia necessario utilizzare altre variabili oltre i parametri formali, queste vanno dichiarate; poiché si fa riferimento ad aree diverse di memoria è però

consentito, anche se non consigliato, dichiarare variabili che hanno lo stesso nome di quelle presenti nel main o in altre funzioni, senza che vi sia alcun tipo di errore.

Ad ogni funzione è associato un tipo di "valore di ritorno", che può essere un tipo di dato fondamentale, derivato o `void` se la funzione non deve restituire alcun valore.

Il concetto di "restituzione" di un valore da parte di una funzione consiste nel memorizzare il valore di una variabile o di una espressione presente nella funzione, in una variabile presente nel main. Il valore di ritorno è restituito dalla funzione mediante l'istruzione `return`. Esempio:

```
int cubo(int c)
{
        int d;
        d=c*c*c;
        return d;
}
```

in questo caso la funzione "cubo" calcola e restituisce il valore della variabile intera d che verrà assegnato ad una variabile nel main.

3.3 Chiamata di una funzione

Quando nel main si ha il bisogno di chiamare la funzione, lo si fa facendo riferimento al nome e passando a essa una lista di parametri conforme in tipo, numero e ordine alla lista dei parametri elencati nella dichiarazione della funzione, detti "parametri attuali":

```
var_di_ritorno = nome_funz(val1,val2,…,valn);
```

o se essa non deve restituire nessun tipo di dato:

```
nome_funz(val1,val2,...,valn);
```

Si noti che, conseguentemente a quanto detto nel paragrafo precedente circa la diversa area di memoria a cui fanno riferimento i blocchi istruzione delle funzioni, le variabili passate come argomento contengono i valori in ingresso di quella specifica chiamata di funzione che vengono memorizzati nei rispettivi "parametri formali", dunque i "paratetri attuali" non vengono in alcun modo modificati (scopriremo che in alcuni casi questo può rappresentare un limite, che verrà superato con l'introduzione dei "puntatori"). In C infatti il passaggio dei parametri avviene sempre e soltanto per valore, ecco perché deve esistere una coerenza di tipo e di numero tra parametri formali e parametri attuali. Esempio:

```
void main()
{
        int a, b;
        scanf("%d",&a);
        b=cubo(a);
```

```
            printf("%d al cubo vale %d",a,b);
}
```

in questo esempio viene chiamata la funzione "cubo", passandole il valore precedentemente acquisito della variabile "a", il valore restituito dalla funzione viene assegnato a "b" e poi stampato a schermo.

4 Puntatori

Un puntatore è un tipo speciale di variabile che contiene l'indirizzo di memoria di un'altra variabile e attraverso l'operatore * detto "operatore di indirezione", applicabile solo ad una variabile puntatore, restituisce il contenuto dell'oggetto puntato.

La sintassi di definizione di una variabile puntatore è la seguente:

```
tipo_base *var_punt;
```

dove `var_punt` è definita come variabile di tipo "puntatore a `tipo_base`"; `tipo_base` può essere uno dei tipi di dati fondamentali già introdotti, int, char, float, double, oppure un tipo di dato strutturato dichiarato in precedenza. Esempio:

```
int a;
int *p;
p=&a;
*p=a;
```

in queste linee di codice dopo l'assegnazione `p=&a`, i nomi `a` e `*p` sono perfettamente equivalenti in quanto permettono di accedere agli stessi byte in memoria: infatti in p viene memorizzato l'indirizzo della variabile a e l'operatore * anteposto a p permette a quest'ultimo di puntare al contenuto di a.

4.1 Puntatori e vettori

Il concetto di puntatore è estendibile anche ai vettori, poiché in C è possibile agire su un puntatore come se si stesse agendo su un vettore, e viceversa:

```
int i, vett[10];
int *p;
...; p=vett;
for(i=0;i<10;i++)
            printf("%d",p[i]);
```

con queste istruzioni si memorizza in `p` l'indirizzo della prima componente del vettore `vett` e attraverso l'indice tra parentesi quadre `i`, come per i vettori, si fa puntare `p` al contenuto dell'i-esimo elemento di `vett`.

4.2 Puntatori e funzioni

Come accennato nel paragrafo 4.3, i puntatori risultano molto utili se utilizzati con le funzioni.

Generalmente vengono passati alle funzioni le variabili per valore (utilizzando return). Ma questo modo di passare gli argomenti non modifica gli argomenti stessi e quindi potrebbe risultare limitativo quando, invece, vogliamo che vengano modificate le variabili che passiamo alle funzioni.

Per far in modo che una funzione modifichi gli argomenti passati ad essa, si passano alla funzione non i valori delle variabili, ma il loro indirizzo, trovandoci, quindi, ad operare con i puntatori a tali valori. Di conseguenza quando la funzione è chiamata nel main e vengono passati gli indirizzi delle variabili come argomenti, in corrispondenza i parametri formali devono essere dichiarati di tipo puntatore, in modo che possano puntare al contenuto delle suddette variabili nel main. Esempio:

```
void cubo(int*);
void main()
{
          int a;
          scanf("%d",&a);
          cubo(&a);
          printf("Il cubo del valore immesso e' %d\n",a);
}
void cubo(int*A)
{
          *A=(*A)*(*A)*(*A);
}
```

Ringraziamenti. Il presente capitolo è stato scritto anche grazie al prezioso contributo degli studenti Donato Mancuso, Giuseppe Ragno, Flavio Palmieri, Pasquale Bonasia.

Riferimenti

1. Bevilacqua, V.: Dispense Linguaggio C In: http://www.vitoantoniobevilacqua.it
2. http://it.wikipedia.org/wiki/Ricerca_binaria
3. http://it.wikipedia.org/wiki/Stringa_(informatica)
4. http://it.wikipedia.org/wiki/Struttura_dati
5. http://it.wikipedia.org/wiki/Funzione_(informatica)
6. http://it.wikipedia.org/wiki/Puntatore_(programmazione)

Appendice: Codice in linguaggio C

```
//ESERCIZIO: ORDINA STUDENTI
#include <stdio.h>
/*dichiarazione dato strutturato*/
struct studente
{
  char cognome[20];
  char nome[20];
  int voto;
};
/*dichiarazione funzioni con argomenti puntatori*/
void leggi(struct studente*,int);
int ordina(struct studente*,int);
void stampa(struct studente*,int);
void scambia(struct studente*,struct studente*);
void main()
{
  int N;
  struct studente classe[100];
  do
  {
      printf("Inserisci il numero di studenti\n");
      scanf("%d",&N);
  }
  while(N>100);
  leggi(classe,N);
  printf("Gli studenti che compongono la classe sono\n");
  stampa(classe,N);
  N=ordina(classe,N);
  printf("%d studente/i passato/i:\n",N);
  stampa(classe,N);
}
/*definizione funzioni*/
void leggi(struct studente*p,int N)
{
  int i;
  for(i=0;i<N;i++)
  {
      printf("%d^ studente: cognome,nome,voto\n",i+1);
      scanf("%s %s %d",p[i].cognome,p[i].nome,&p[i].voto);
  }
}
int ordina(struct studente*p,int N)
{
  int imax,i,j;
  for(i=0;i<N-1;i++)
  {
      imax=i;
      for(j=i+1;j<N;j++)
            if(p[j].voto>p[imax].voto)
                  imax=j;
      scambia(&p[i],&p[imax]);
  }
  i=0;
```

```
  while(p[i].voto>=18)
  {
      if(p[i].cognome[0]>=97 && p[i].cognome[0]<=122)
            p[i].cognome[0]=p[i].cognome[0]-32;
      if(p[i].nome[0]>=97 && p[i].nome[0]<=122)
            p[i].nome[0]=p[i].nome[0]-32;
      i++;
  }
  return i;
}
void scambia(struct studente*p,struct studente*q)
{
  struct studente temp;
  temp=*p;
  *p=*q;
  *q=temp;
}
void stampa(struct studente*p,int A)
{
  int i;
  for(i=0;i<A;i++)
      printf("%s %s %d\n",p[i].cognome,p[i].nome,p[i].voto);
}
```