C array di caratteri

Il tipo char ha dimensione di un byte e contiene il numero di codice ASCII che gli corrisponde.

Un letterale di tipo carattere deve essere sempre racchiuso tra virgolette semplici, ad esempio: ‘a’.

Oltre agli array di numeri interi o reali, possiamo definire anche gli array di caratteri; per esempio

```
char s[]={‘c’, ‘i’, ‘a’, ‘o’}
```

dove s è un array di caratteri formato da 4 elementi

```
s[0]=’c’; s[1]=’i’; s[2]=’a’; s[3]=’o’;
```

Una **stringa** è un **insieme di caratteri** (e/o sequenze di escape) rappresentate tra virgolette:

```
"Hello World\n"
```

Le stringhe in C non costituiscono un tipo standard

non sono ammesse come operandi dalla maggior parte degli operatori (compreso l'operatore di assegnazione)

Sono tuttavia riconosciute da alcuni operatori (di flusso di I/O) e da numerose funzioni di libreria del C (di manipolazione stringhe, come `printf`)

Per gestire un array di caratteri come stringa non è necessario conoscere a priori la sua lunghezza in quanto è sufficiente scandire la stringa fino al simbolo '\0'.

C array di caratteri

In memoria le stringhe sono degli array di tipo char null terminated …

… array di caratteri con la particolarità che l'elemento che segue l'ultimo carattere deve contenere il carattere NULL (`'\0'` terminatore)

Consente a operatori e funzioni di distinguere stringhe da normali array

```
char s1[] = {'h','e','l','l','o'}
char s2[] = {'h','e','l','l','o','\0'}
char s3[] = "hello";
```

s1 non è una stringa, ma un array di caratteri
s2 e s3 sono stringhe e sono uguali)

Esempio: Leggere una stringa

```
int main()
{
   char s[30];

   printf("inserisci la stringa”);
   scanf("%s”,s); // Notare che non serve &

   for(int i=0;i<30;i++)
    printf(”%c”,s[i]);

   printf(“\n”);
   printf(“%s”,s);
   return 0;
}
```

Se non viene fornito un array di caratteri abbastanza grande per memorizzare le stringhe in input potreste perdere dei dati o incorrere in errori gravi durante l’esecuzione del programma.

Usando `scanf` per la gestione dell'input, la lettura di ciascuna variabile termina quando incontra o il carattere spazio (la barra spaziatrice) o la tabulazione (tasto *Tab*) o il fine linea (tasto *Invio*).

ESEMPIO

Scrivere un programma che, dato un nome in input, costruisca una stringa contenente il testo

ciao, *nome*

Esempio: se l'utente digita il nome *Antonio*, il programma deve restituire la stringa str contenente il testo

ciao, *Antonio*

I passi da seguire sono i seguenti:

Poni la scritta "ciao," nella variabile str;
Leggi(nome);
Aggiungi tutti i carattere del nome
Alla fine aggiungi il carattere nullo
Stampa la stringa str;

Ci serviamo di un indice **j** che scorre tutti i caratteri della stringa **nome** assegnata in input finché non raggiunge il carattere nullo; ognuno di questi caratteri deve essere aggiunto alla stringa ***str*** dopo "**ciao**, ", cioè dall'indice 6 in poi.

Esempio: scrivere un programma che, dato un `nome` in input, costruisca e stampi una stringa con il testo `"ciao, nome"`

```
int main()
{
    char nome[20];
    char str[30]="ciao, "; // str contiene 6 caratteri + '\0'
    int j=0;
    printf("nome: ");
    scanf("%s",nome);
    while (nome[j]!='\0')
        str[j+6]=nome[j++];

    str[j+6]='\0';
    printf("%s",str);
    return 0;
}
```

Esercizi:

1. Assegnata una stringa `s`, scrivere un programma che conteggi i caratteri `'a'` in essa contenuti.
2. Assegnata una stringa `s`, ed un intero `n` minore della lunghezza della stringa, scrivere un programma che stampi tutti i caratteri compresi tra `n` e la lunghezza della stringa.
3. Assegnata una stringa `s`, e due interi `m<n` minori della lunghezza della stringa, scrivere un programma che stampi la sottostringa che va da `m` ad `n`

   Esempio: `s="ciao mamma"` ed `m=3`, `n=4` il programma deve stampare la stringa `"ao ma"`
4. Scrivere un programma che, assegnata in input una stringa `s` del tipo `aaa@bbb` stampi su video

```
nome: aaa
indirizzo: bbb
```

Esercizi:

1. Scrivere un programma che data in input una stringa la compatti in una nuova stringa eliminando tutti gli spazi in essa presenti
2. Scrivere un programma che data in input una stringa (testo) stampi i caratteri distinti (inclusi caratteri di punteggiatura e spazi) in essa presenti e quante occorrenze di ciascuno di essi si trovano nella stringa
3. Scrivere un programma che date in input due stringhe *A* e *B* fornisca la posizione in *A* (indice) di tutte le occorrenza di *B*
   - Esempio: A="In questo testo cercare le occorrenze della parola \"esto\"";

     B="esto" posizione=5,11,52

Esercizio. ***Assegnata una stringa s contenente anche delle cifre, sommare tutte le cifre e fornire il valore totale di tale somma.***

Esempio, : s="abc3x casa2 y34zq" il programma deve restituire 3+2+3+4=12.

Pseudo codice:

assegnare la stringa *s*;
scandire ogni carattere *c* delle stringa finché non termina; (*while*)
verificare se è compreso tra 0 e 9
se lo è incrementare la somma con il valore corrispondente della cifra;
passare al carattere successivo;
stampare la somma totale.

Nel ciclo ***while*** dobbiamo specificare

inizializzazione: contatore di ciclo inizializzato a zero, **i = 0**, e **somma = 0**;

condizione di uscita: l'ultimo carattere letto è la sequenza di escape \0 **c == '\0'**; la condizione di esecuzione del ciclo è la sua negazione, **c != '\0'**;

corpo del ciclo: se il carattere è una cifra aggiungerla a somma; incrementare in ogni caso il contatore **i**;

Osserviamo che l'istruzione ***(int) c*** effettua un casting esplicito da carattere a numero: ciò che si ottiene è il codice ASCII di ***c***; se sottraiamo tale valore al codice ASCII del carattere che rappresenta lo zero, otteniamo proprio il valore numerico della cifra. In altre parole,

(int) '3' – (int) '0' ***restituisce l'intero 3,*** ***mentre*** ***(int) c – (int) '0'*** ***restituisce l'intero rappresentato dal carattere c, se c è compreso tra 0 e 9.***

Esempio: assegnata una stringa *s* contenente anche delle cifre, sommare tutte le cifre e fornire il valore totale di tale somma.

Esempio: `s="abc3x casa2 y34zq"` il programma deve restituire 3+2+3+4=12.

Esempio: assegnata una stringa *s* contenente anche delle cifre, sommare tutte le cifre e fornire il valore totale di tale somma.

Esempio: `s="abc3x casa2 y34zq"` il programma deve restituire 3+2+3+4=12.

```
int main()  {
    char s[100];
    int i=0, somma=0;
    printf("Stringa=”);


    while (s[i]!= '\0') {
        if ((s[i]>='0') && (s[i]<='9'))
            somma+=(int)s[i]-(int)'0';
        i++;
    }
    cout<<"somma="<<somma<<endl;
    system("pause");
}
```

C tipi astratti

il C consente al programmatore di definire nuovi tipi, estendendo così le capacità effettive del linguaggio

… una volta definiti, questi tipi sono molto "concreti" e sono trattati esattamente come i tipi standard.

I tipi non standard sono detti **tipi astratti**:

Tipi enumerativi on **`enum`**

Nuovi tipi con **`typedef`**

Strutture (e puntatori a strutture) (modulo B)

Oggetti e classi (modulo B)

C tipi astratti -> enumerativo

Con la parola-chiave **`enum`** si definiscono i **tipi enumerati**

Variabili di tipi enumerati possono assumere solo i valori specificati in un elenco

```
enum Nome-Tipo { lista dei valori accettabili }
```

Il tipo enumerativo ha le seguenti caratteristiche:

1. è un tipo definito dal programmatore che può essere utilizzato nella dichiarazione di una variabile
2. la lista dei valori accettabili è un insieme di identificatori
   - rappresentano i soli valori che le variabili di quel tipo possono assumere;
   - non si può assegnare a variabili di quel tipo valori diversi da quelli prescritti:
3. è un tipo ordinale nel senso che si può parlare di un precedente (tranne il primo) e di un successivo (tranne l'ultimo);
4. all'interno della macchina il dato viene trattato come un intero

C tipi astratti -> enumerativo

Il C associa ad ogni identificatore definito nella lista un numero intero che parte da zero; perciò, lunedì corrisponde a 0, martedì ad 1, e così via (non sono accettate le vocali accentate).

Ha sicuramente senso scrivere

oggi <= domani

il compilatore interpreterà il risultato sostituendo i rispettivi valori alle variabili e restituirà un valore vero o falso.

Supponiamo invece di scrivere

domani < = ris

In fase di compilazione il compilatore dovrebbe dare un errore.

Invece, il migliore dei compilatori fornisce un errore di avvertimento (warning)!

C tipi astratti -> enumerativo

Agli enumeratori sono assegnati numeri interi, a partire da 0 e con incrementi di 1

Volendo assegnare numeri diversi (comunque sempre interi), bisogna specificarlo.

Esempio:

```
enum feriale { Lun, Mar, Mer=4, Gio, Ven };
```

l'uso degli enumeratori, anziché quello diretto delle costanti numeriche corrispondenti, è utile in quanto permette di scrivere codice più chiaro ed esplicativo di ciò che si vuole fare.

Esempi:

```
enum giorno { Lun, Mar, Mer, Gio, Ven, Sab, Dom };
enum risposta {si, no, nonso};
giorno oggi, domani;
risposta ris;
oggi = Mer;
oggi = si;
...
if (oggi <= domani) ...
if (ogg <= ris) ...
```

Il tipo enumerativo viene usato nella dichiarazione di variabili

Errore!: la variabile di tipo enumerato non può assumere valori diversi da quelli prescritti

Il tipo enumerativo è ordinale

Warning!: confronto di variabili di tipo enumerativo diverso

C tipi astratti -> enumerativo

Consideriamo ancora la possibilità di utilizzare un costrutto iterativo per evidenziare tutti i giorni della settimana:

for (oggi=lunedì; oggi <= domenica; oggi++)
{ …….istruzioni…… }

il compilatore evidenzia un errore!

Infatti, poiché ***oggi ++*** è equivalente a ***oggi = oggi+1***

il compilatore forza il valore ***oggi*** a destra dell'uguaglianza a diventare un intero, per poi assegnare alla variabile ***oggi*** (che appare alla sinistra) ancora un intero;
in questo modo la variabile ***oggi*** perde il suo significato iniziale per diventare un intero.

Si può ovviare a questo inconveniente forzando con un ***casting*** al tipo ***giorno*** il risultato.

for (oggi=lunedì; oggi <= domenica; oggi=(giorno)(oggi+1))

il valore di ***oggi+1*** è forzato ad assumere un valore di tipo giorno.

Esempio

```
int main() {
enum giorno {lunedi=1, martedi, mercoledi, giovedi, venerdi, sabato,
   domenica};
enum giorno oggi;

printf("Giorni della settimana\n");

for (oggi=lunedi; oggi<=domenica; oggi=(enum giorno)oggi+1)   {
   printf("%d  ", oggi);
   if (oggi<=venerdi)
        printf(" giorno lavorativo\n");
   else if (oggi==sabato)
        printf(" giorno semifestivo\n");
   else                    printf(" giorno festivo\n");
}
return 0;
}
```

C tipi astratti -> typedef

L'istruzione **`typedef`** serve a dare un nuovo nome (identificatore) ad un tipo (*standard* o *astratto*) già esistente

`typedef` non crea un nuovo tipo

Esempi:

```
typedef unsigned int Uint;
```

crea il tipo `Uint` che corrisponde all'intero senza segno

```
typedef int Integer;
```

crea il nuovo nome `Integer` per il tipo standard **`int`**

C tipi astratti -> typedef

Un motivo per cui si utilizza **`typedef`** è la **portabilità del software**

- Confinando in unico luogo i riferimenti diretti a un tipo è più semplice scrivere programmi portabili su macchine diverse
- Esempio: dobbiamo definire un tipo intero a 32 bit, portabile su diverse macchine:

```
...
#ifdef BIT32MACHINE
typedef int int32;
#else
typedef long int32;
#endif
...
```

- Ovviamente tutte le definizioni di interi devono essere sostituite nel programma con il tipo `int32`

C tipi astratti -> typedef

Un altro motivo per cui si utilizza **typedef** è per evitare ripetizioni di definizioni lunghe di tipi

Esempio: dobbiamo definire più array dello stesso tipo-componente e dimensione:

```
...
double a1[100];
double a2[100];
...
```

può essere sostituito da:

```
typedef double acento[100];
acento a1, a2;
...
```