Nozioni sulle cstringhe

Gabriele Frassi

Indice

1 Nozioni base

Premessa In C++ non esiste il tripo stringa!

Una *cstringa* consiste in una sequenza di caratteri avente lunghezza arbitraria, cioè in un qualunque array di caratteri che contiene, ad un certo punto, il carattere di arresto $\backslash 0$. Ricordiamo che posso esprimere un letterale stringa racchiudendo una sequenza di char tra virgolette. Per esempio ho *char* $c[5]$, che consiste in una cstringa che può contenere una parola avente al più 4 caratteri, tenendo conto della presenza del carattere di arresto. All'interno di una cstringa posso avere altri caratteri speciali come quello di ritorno a capo $\backslash n$ o di tabulazione $\backslash t$.
Una cstringa può essere inizializzata fondamentalmente in due modi:

- come un qualunque array: $char\ carray1[] = \{'C','a','l','c','o','l','o','\backslash 0'\}$
- ricorrendo a un letterale stringa: $char\ carray2[] = "Calcolo"$;

Nella prima inizializzazione è necessario indicare il carattere di arresto, nella seconda viene incluso automaticamente. Ricordiamo che negli array non è definito l'assegnamento. L'uso di un letterale stringa è pertanto concesso solo nell'inizializzazione: non potremo porre, per esempio, $variabile1 = variabile2$.

Gli operatori di ingresso e di uscita accettano una variabile stringa come argomento.

L'operatore di ingresso legge caratteri dallo stream di ingresso (salta eventuali spazi bianchi di testa) e si ferma dopo aver incontrato uno spazio bianco. Questo carattere, che non viene letto, provoca il termine dell'operazione di lettura e la memorizzazione nella variabile stringa con in fondo (dopo l'ultimo carattere letto) il carattere di arresto: tutto questo avviene, ovviamente, avendo una variabile stringa di adeguate dimensioni.

L'operatore di uscita , invece, scrive i caratteri della stringa (ad eccezione del carattere di arresto) sullo stream di uscita.

2 Libreria $< cstring >$

Risulta necessario, non potendo utilizzare operazioni di assegnamento e/o operatori di confronto, introdurre delle funzioni aggiuntive per manipolare le nostre cstringhe. Le funzioni in questione si trovano nella libreria $< cstring >$ e consistono nelle seguenti:

- **strlen**, restituisce un intero che consiste nel numero di caratteri (ad eccezione del carattere di arresto) contenuti nella cstringa.
- **strcpy**, funge da surrogato all'operatore assegnamento e permette di copiare il valore di una cstringa *source* in una cstringa *destination*
- **strcmp**, funge da surrogato a operatori di confronto e permette di confrontare due cstringhe. Viene restituito un intero che può essere 0, positivo o negativo:
  - se viene restituito 0 le due cstringhe hanno lo stesso valore
  - se viene restituito un numero positivo significa che il primo carattere diverso, da sinistra verso destra, ha un valore più grande nella prima cstringa rispetto alla seconda (**Es**: se $str1 =' Ciao'$ ed $str2 =' Addio'$ allora $strcmp(str1, str2) > 0$)
  - se viene restituito un numero negativo significa che il primo carattere diverso, da sinistra verso destra, ha un valore più piccolo nella prima cstringa rispetto alla seconda (**Es**: se $str1 =' Addio'$ ed $str2 =' Ciao'$ allora $strcmp(str1, str2) < 0$)
  - se ho due stringe aventi lunghezza diversa ma con corrispondenza tra tutti i caratteri:
    - se il carattere di arresto viene incontrato primma nella $str1$ restituisco -1
    - se il carattere di arresto viene incontratoprima nella $str2$ restituisco 1
- **tolower**, sfruttando le operazioni di addizione e sottrazione delle stringhe rendo ogni carattere maiuscolo della cstringa minuscolo
- **tolupper**, sfruttando gli stessi meccanismi di tolower rendo ogni carattere minuscolo della cstringa maiuscolo
- **strcat**, concateno due cstringhe in una sola.

2.1 strlen

Immaginiamo di costruire una nostra *my_strlen*:

```
int my_strlen(const char* str) {
        int i = 0;
        while(str[i] != '\0') {
                i++;
        }
        return i;
}
```

Come argomento abbiamo un puntatore a cstringa. Il puntatore viene depotenziato per evitare errori di modifica.

2.2 strcpy

Immaginiamo di costruire una nostra *my_strcpy*:

```
void my_strcpy(char* d, const char* s) {
        int i = 0;
        while(s[i] != '\0') {
                d[i] = s[i];
                i++;
        }
        d[i] = '\0';
}
```

Prima di chiamare funzione risulta estremamente utile utilizzare la *strlen*: non posso copiare il contenuto di una cstringa di una certa dimensione in una avente dimensione minore.

2.3 strcmp

Immaginiamo di costruire una nostra *my_strcmp*:

```
int my_strcmp(const char* s1, const char* s2) {
        int i = 0;
        while(s1[i] != '\0' && s2[i] != '\0') {
                if(s1[i] < s2[i])
                        return -1;
                else if(s1[i] > s2[i])
                        return 1;
                i++;
        }

        if(s1[i] != '\0' && s2[i] == '\0')
                return 1;
        if(s1[i] == '\0' && s2[i] != '\0')
                return -1;

        return 0;
}
```

Nella funzione viene svolto un confronto carattere per carattere. Il ciclo si interrompe quando viene trovata la prima differenza e si restituisce un valore positivo o un valore negativo in base a quanto detto nell'introduzione. Nel caso in cui venga incontrato il carattere di arresto in una delle due cstringhe, restituisco valore positivo se la cstringa $s2$ risulta più corta della cstringa $s1$, valore negativo se la cstringa $s1$ risulta più lunga della cstringa $s2$

2.4 tolower

Immaginiamo di costruire una nostra *my_tolower*:

```
void my_tolower(char* s) {
        for(int i =0; i < strlen(s); i++) {
                if(s[i] >= 'A' && s[i] <= 'Z')
                    s[i] = s[i] - 'A'+'a';
        }
}
```

2.5 tolupper

Immaginiamo di costruire una nostra *my_tolupper*:

```
void my_tolupper(char* s) {
        for(int i =0; i < strlen(s); i++) {
                if(s[i] >= 'a' && s[i] <= 'z')
                    s[i] = s[i] - 'a'+'A';
        }
}
```

2.6 strcat

Cococcioni non ci ha indicato un codice preciso come con le altre funzioni. **Nozioni**:

- Concatenare, come già evidenziato introducendo i letterali stringa, consiste unire due letterali stringa in un unico literal. Il valore di questo è l'unione dei valori dei due literal divisi.
- Mediante la funzione di libreria *strlen* individuo la lunghezza della cstringa. Ricordare che l'strlen non considera, nella somma, il carattere di arresto.

  Esempio $n = strlen(ch1) + strlen(ch2) + 1$.

- Mediante un primo ciclo inserisco nella nuova cstringa i caratteri di *ch1*, ignorando il carattere di arresto.
- Dopo aver stabilito il valore di i elementi della nuova cstringa includo, mediante un ulteriore ciclo, i caratteri di *ch2*. In questo ciclo partirò dalla posizione i (avendo occupato gli elementi da indice 0 a indice $i-1$). Il codice del ciclo non cambia (cambia solo la posizione da cui si parte)
- Includo alla fine il carattere di arresto $\backslash 0$.