

Dal Pascal al C

Parte 5^a

Aritmetica dei puntatori (1)

Nel C è possibile usare vari operatori su puntatori. Se **p** e **q** sono due variabili dello stesso tipo puntatore ed **i** è un valore di tipo intero sono sintatticamente corrette le seguenti espressioni: **p+i**, **p-i**, **p++**, **++p**, **p--**, **--p**, **p+=i**, **p -=i**, **p==q**, **p!=q**, **p < q**, **p <=q**, **p >q**, **p >=q**.

```
typedef struct {  
    int giorno;  
    int mese;  
    int anno;  
} data;  
  
data ricorrenze[100];  
data *p, *q;
```

Nell'esempio a lato, si definiscono l'**array ricorrenze** di **100 strutture** di tipo **data** e due **puntatori**, **p** e **q**, di tipo **puntatore a data**. (Si ricorda che nel C gli elementi di un array sono memorizzati in locazioni contigue di memoria)

Aritmetica dei puntatori (2)

Le tre seguenti sequenze di istruzioni hanno lo stesso effetto.

```
printf("giorno %d", ricorrenze[6].giorno);
```

```
p=&ricorrenze[4];  
p=p+2;  
printf("giorno %d", (*p).giorno);
```

```
p=&ricorrenze[4];  
p=p++;  
p=p++;  
printf("giorno %d", (*p).giorno);
```

Nel secondo frammento di codice prima si assegna a **p** l'indirizzo del **quinto** elemento (una struttura) dell'array **ricorrenze**; poi si aggiunge **2**; quindi si utilizza il nuovo indirizzo per accedere al **settimo** elemento dell'array.

L'incremento dell'indirizzo (**p=p+2**) non ha il semplice effetto di sommare l'intero **2** a **p**, ma quello di sommare a **p** un numero pari al doppio del numero di bytes della struttura di tipo **data**.

Nel terzo frammento di codice viene usato l'operatore **++** al posto dell'operatore **somma**. In generale, l'incremento di una variabile puntatore avviene in funzione del tipo dell'elemento puntato. Anche nel caso di differenza tra puntatori, il risultato dall'espressione rappresenta il numero di elementi tra i due puntatori.

Puntatori ed array (1)

Il fatto che nel C gli elementi di un array sono memorizzati in locazioni di memoria contigue di memoria e le considerazioni, appena viste, sull'aritmetica dei puntatori, suggeriscono una interpretazione dell'indirizzamento di un array molto legata alla implementazione. Ad esempio sia v un array monodimensionale di interi e consideriamo l'espressione $v[4]$. Possiamo pensare che v sia un puntatore al primo elemento dell'array e che $v[4]$ sia l'equivalente di $*(v+4)$, cioè che $v[4]$ che non sia altro che una notazione compatta per denotare l'elemento puntato da un puntatore dopo uno spostamento di 4 posizioni in avanti.

Questa interpretazione del tipo array è perfettamente supportata dal linguaggio C.

Puntatori ed array (2)

```
int v[10];
int i;
i=0;
while (i<=9) {
    printf(“%d”, v[i]);
    i++;
}
```

```
int v[10];
int *p;
p=v;
while (p<=&v[9]) {
    printf(“%d”, *p);
    p++;
}
```

I due frammenti di programma riportati a lato sono equivalenti; entrambi scandiscono un array e ne stampano gli elementi. Il frammento di programma “più tradizionale” ha nel C una versione equivalente in cui l’identificatore dell’array viene visto come un puntatore.

L’assegnazione **p=v**, equivalente a **p=&v[0]**, è sintatticamente corretta proprio perché l’identificatore di un array nel C non è altro che un puntatore allo stesso tipo di quello degli elementi dell’array.

Si noti che **v=p** sarebbe una istruzione non lecita; infatti nel C l’identificatore di un array è sì un puntatore ma non è possibile modificare il suo valore.

Quindi è chiaro come, se **p** è una variabile di tipo puntatore, sia lecito scrivere **p[4]** al posto di ***(p+4)**.

Puntatori e stringhe

Il rapporto esistente tra puntatori ed array rende anche possibile manipolare stringhe attraverso puntatori.

```
ncar = 0;  
while (*p!='\0') {  
    ncar++;  
    p++;  
}
```

Nel frammento di codice a lato, **p** è un puntatore a carattere e **ncar** è un intero.

Al termine del ciclo **while** la variabile **ncar** contiene il numero di caratteri della stringa il cui primo elemento è puntato da **p**.

Per contare i caratteri della stringa si può anche utilizzare un ciclo **for**, come nel seguente esempio.

```
for (ncar=0; *p!='\0'; p++)  
    ncar++;
```

Direttive di compilazione

Dal punto di vista sintattico una direttiva in C si presenta come un comando preceduto dal carattere **#** (ad es. **#define N 100**).

Le direttive vengono eseguite dal compilatore prima di ogni altra attività, in una fase che usualmente è chiamata di pre-compilazione. Una direttiva ha come obiettivo quello di modificare il testo del programma prima che il programma venga compilato. Ad esempio, quando nella pre-compilazione viene incontrata la direttiva **#define N 100**, da quel punto in poi, nel testo del programma l'identificatore **N** viene sostituito dalla costante **100**.

La direttiva **include**

La sintassi della direttiva **include** è la seguente:

#include *nome-file*

Quando viene incontrata una direttiva **include**, la direttiva è rimpiazzata, nel codice del programma, dal contenuto del file con nome ***nome-file***; ***nome-file*** può essere racchiuso o tra < e > oppure tra doppi apici, .

Se si utilizza la forma con i simboli < e >, il file viene cercato nella directory che contiene i ***file header standard*** del C; il nome di questa directory varia da compilatore a compilatore e da installazione a installazione.

Se si utilizza la forma “***nome-file***” in cui il ***nome-file*** è racchiuso tra doppi apici, allora il file viene cercato nella directory corrente.

Le direttive di definizione (1)

La direttiva **define**, che abbiamo già utilizzato per definire le costanti, ha una forma sintattica più generale che è la seguente:

#define *identificatore specifica-definizione*

specifica-definizione può essere o una **costante**, o (***lista-identificatori***) ***sequenza caratteri***.

Nella versione con le parentesi, la direttiva **define** permette di definire delle **macro**, i cui parametri sono elencati nella lista di identificatori tra parentesi. Il corpo della macro è costituito da una sequenza di caratteri. Ecco un esempio di macro: **#define minimo(x,y) ((x)>(y) ? y : x)**

Dopo aver incontrato questa direttiva, se ad esempio il pre-compilatore incontra nel codice la sequenza di caratteri **minimo(a+b,c)** la sostituisce con la sequenza **((a+b)>(c) ? c : a+b)**.

? : è l'operatore condizionale, esso valuta l'espressione che precede **?**, se l'espressione restituisce un valore diverso da **0** (espressione **vera**) allora viene valutata l'espressione compresa tra **?** e **:** e l'espressione condizionale restituisce il risultato di tale valutazione; altrimenti (espressione **falsa**) viene valutata l'espressione che segue il simbolo **:** e l'espressione condizionale restituisce il risultato di tale valutazione. Ad es. **a>0 ? a : -a**, restituisce il valore **a** se **a** è positivo, altrimenti restituisce **-a**.

Le direttive di definizione (2)

Se nel corpo della macro il parametro formale è preceduto da un **#**, allora nella espansione della macro l'identificatore che fa da parametro attuale viene posto tra doppi apici. Se nel corpo della macro un formale è separato da un altro carattere da un doppio cancelletto (**##**), allora nella espansione della macro il parametro attuale non è separato dal carattere da nessuno spazio.

Nel C è disponibile anche la direttiva **undef**, con la seguente sintassi:

#undef identificatore

Quando il pre-compilatore incontra la direttiva **undef**, se l'identificatore specificato era stato definito in precedenza tramite la direttiva **define**, tale definizione non viene più considerata valida.

Compilazione condizionale

La direttiva **ifdef** consente di effettuare la compilazione di una porzione di codice piuttosto che un'altra con la seguente sintassi.

#ifdef identificatore sequenza-caratteri

#else sequenza-caratteri (può non essere presente)

#endif

Se l'identificatore è stato definito viene compilata la prima sequenza di caratteri; se è presente la parte **else** e l'identificatore non è stato definito viene compilata la seconda sequenza di caratteri (quella dopo l'**else**).

La direttiva **ifndef** è analoga sintatticamente alla **ifdef** ma ha un comportamento opposto: viene compilata la prima sequenza se l'identificatore non è stato definito.

Aprire il file **stdio.h** che si trova nella cartella **Include** del Dev C++ per vedere esempi di **ifdef** e **ifndef**.

Librerie standard del C

Le funzioni di libreria sono già compilate e usualmente i file oggetto hanno estensione **.lib** e sono disponibili nella directory di nome **lib**.

Per usare una funzione compilata separatamente è necessario specificare nel file da compilare il prototipo della funzione.

Nel caso delle librerie standard il C mette a disposizione, per comodità del programmatore, un insieme di file in cui sono memorizzati i prototipi delle funzioni di libreria, raggruppati per funzioni omogenee. Ad esempio il file **stdio.h** contiene tutti i prototipi delle funzioni di **input-output**. I file che contengono prototipi di funzione si chiamano di solito file **header**, hanno estensione **.h** e sono memorizzati nella directory di nome **Include**.

Per usare una funzione di libreria è sufficiente, invece di descrivere esplicitamente il prototipo della funzione da utilizzare, includere nel file da compilare il file **header** opportuno. Ad esempio per usare le funzioni di **input-output** possiamo premettere al programma la direttiva **#include<stdio.h>**.

Le funzioni per le stringhe

Per la manipolazione delle stringhe il C mette a disposizione diverse funzioni di libreria. Il file **header** contenente i prototipi è **string.h**. Se non è specificato il contrario, si intende che le stringhe terminano con il carattere **'\0'** di fine riga.

Le funzioni matematiche

Le librerie standard del C offrono un insieme di funzioni matematiche molto ampio, che spaziano dalle usuali funzioni trigonometriche e logaritmiche, alle funzioni iperboliche, fino al calcolo delle funzioni di Bessel. Il file **header** corrispondente è **math.h**.

Le funzioni per i caratteri

Il linguaggio C offre un vasto insieme di funzioni per testare e manipolare caratteri; queste funzioni, definite nel file **header ctype.h**, permettono di classificare i caratteri in cifre, caratteri alfanumerici, spazi, maiuscole, minuscole e così via: Tutte queste funzioni accettano come parametri di ingresso e restituiscono interi, in accordo con le regole di conversione implicita del linguaggio.

Le funzioni di conversione

Nel C esistono delle funzioni che permettono di convertire stringhe in numeri. Queste funzioni sono presenti nel file **header stdlib.h**.

Le funzioni di input-output

Il linguaggio C mette a disposizione varie funzioni, dichiarate nel file **header `stdio.h`**, per la gestione dell'input-output a vari livelli.

Tutti i dispositivi di input ed output sono trattati dal linguaggio C allo stesso modo, ossia come dispositivi in cui è possibile scrivere o leggere un flusso (**stream**) di dati. Esistono tre flussi definiti dallo standard, che di solito sono associati al terminale: **stdin**, **stdout** e **stderr**. Il primo è il flusso standard di input; il secondo è il flusso standard di output; il terzo, che di fatto è un flusso di output, viene utilizzato per la visualizzazione di messaggi di errore.

Distinguiamo quindi le funzioni in base al dispositivo di input-output coinvolto: ci sono funzioni che operano espressamente sui dispositivi di lettura e scrittura (**stdin** e **stdout**) e funzioni che operano su file.

Funzioni che operano su file

```
FILE *fopen(const char *fn, const char *mode);  
int fclose(FILE *pf);
```

La funzione **fopen** apre il file il cui nome e percorso è contenuto nella stringa **fn** e restituisce il puntatore alla struttura di tipo **FILE** associata al file aperto se l'operazione ha avuto successo, **NULL** altrimenti. Il tipo **FILE** è definito in **stdio.h**. Le operazioni sul file sono limitate dalla stringa **mode**.

La funzione **fclose** chiude il file associato al puntatore **pf**. Viene restituito il valore **0** se l'operazione ha avuto successo.

Funzioni per la gestione della memoria

Il C offre varie funzioni per la gestione dinamica della memoria. I prototipi di queste funzioni sono presenti nel file di **header malloc.h**.

Funzioni di ordinamento e ricerca

Nelle librerie del C sono presenti anche due funzioni del tutto generali che permettono di cercare un elemento in un array ordinato (la funzione **bsearch**) e di ordinare un array di qualsiasi tipo (la funzione **qsort**). Il file **header** corrispondente è **stdlib.h**.

La funzione **bsearch()** implementa un algoritmo di ricerca binaria di un elemento in un array.

La funzione **qsort()** implementa l'algoritmo *quick sort* per l'ordinamento di un array.

Funzioni per la generazione di numeri casuali

Come la maggior parte dei linguaggi anche il C ha delle funzioni per la generazione di numeri pseudo-casuali (random). Queste funzioni sono dichiarate nel file **header stdlib.h**.

```
void srand(unsigned int s);  
int rand();
```

La funzione **srand** riceve un intero come parametro di ingresso e lo utilizza per inizializzare il generatore interno di numeri casuali. La funzione **rand** invece restituisce un intero casuale.

Funzioni di interazione con il sistema operativo

Tra le funzioni di libreria sono disponibili **abort**, **exit**, **atexit** e **system** che costituiscono un potente insieme di strumenti per accedere alle funzionalità del sistema operativo.

```
void abort(void);
```

```
void exit(int s);
```

```
int atexit(void(*pfunc)(void));
```

```
void system(const char* s);
```

Le funzioni **abort** ed **exit** provocano la terminazione del programma. La funzione **exit** inoltre provvede alla chiusura di tutti i file ancora aperti ed alla cancellazione dei file temporanei eventualmente creati. La funzione **atexit** riceve in ingresso il puntatore **pfunc** ad una funzione di tipo **void** e senza parametri; questa funzione verrà chiamata automaticamente alla fine dell'esecuzione del programma o in seguito all'esecuzione della funzione **exit**. La funzione **system** permette di specificare nella stringa puntata da **s** un qualunque comando del sistema operativo; tale comando viene eseguito quando la **system** viene invocata.

Riferimenti bibliografici

- J. Welsh J. Elder – Introduzione al Pascal – E.S.A.
- B. W. Kernighan D. M. Ritchie – Linguaggio C – Jackson
- O. Lecarme J.L. Nebut – Pascal – McGraw-Hill
- L. Hancock M. Krieger – Il linguaggio C – McGraw-Hill