di Bruno Rosati

Speciale linguaggi: l'Ansi-C

seconda parte

Dopo l'articolo introduttivo dedicato all'Arm-Assembler ed alle particolarità integrative del Twin, continuiamo il nostro giro di orizzonte sul mondo dei linguaggi carrellando stavolta sulle caratteristiche dell'implementazione dell'Ansi-C sull'Archimedes

Ansi-C

Produttore:
Acorn Computers Cambridge, England

Distributore:
G. Ricordi & C Spa
Via Salomone 77, 20138 Milano

Prezzo *(IVA esclusa): L. 210.000*

Su quello che è, comporta, stabilisce e regolamenta lo standard Ansi — il suo comitato promotore e l'orizzonte di trasportabilità che la standardizzazione ha ulteriormente arricchito — vi è stato spiegato tutto (o quasi...) da Corrado Giustozzi (MC n. 82, pagina 185: Lo Standard Ansi). Sommando a ciò il fatto che dal lontano aprile '87 sulle pagine di MC è presente una rubrica dedicata all'uso del linguaggio e che ancora il medesimo autore vi ha appena invitato a sommergerlo di C-program, non mi sembra proprio il caso di tirar fuori gli stessi strumenti e... suonare la stessa musica.

Così, senza metterci a dire anche noi cos'è il C, quant'è la sua potenza e quanta l'elasticità programmatoria, abbiamo pensato bene a puntare diritto sulle particolarità dell'implementazione e le eccezioni che, rispetto all'Ansi, discostano l'Acorn C-compiler dallo standard.

È questa, praticamente, la stessa impostazione che è stata data all'User Guide anche se, così come le righe d'introduzione della stessa chiaramente consigliano, è soprattutto alla documentazione ufficiale relativa al linguaggio standardizzato che si deve fare riferimento. Sto parlando del «*Draft Proposed American National Standard for Information Systems-Programming Language C*», pubblicato nel 1986. Una sorta di «nuovo testamento» a cui l'User Guide fa solo da raccordo.

Morale della favola: leggete il manuale (tra l'altro non esente da pecche) provate e verificate le eccezioni, ma tenete sempre aperto il «nuovo testamento». Non c'è pagina che non ne invochi la consultazione.

Acorn C-compiler: caratteristiche generali

Il compilatore Acorn, a differenza degli altri sistemi in circolazione, non identifica le proprie classi di file per mezzo dei classici suffissi, bensì, suddivide e cataloga i propri file concentrandoli a seconda del tipo nella relativa directory.

In conseguenza di ciò, tutti i file-sorgente saranno rintracciabili nella subdirectory **«c»**, gli header in quella **«h»**, i vari codici-oggetto nella **«o»**, i programmi nella **«p»**. Da ciò è subito deducibile un'altra particolarità rispetto allo standard. Ovverosia: il mutamento della sintassi degli header che, dal suffisso di riconoscimento convenzionale, ora passano ad essere catalogati con un prefisso.

Per mantenere la piena compatibilità con il mondo esterno, il compilatore traduce automaticamente i nomi degli header rintracciati. Se, come esemplifica lo stesso manuale, si ha a che fare con una linea del tipo:

 # include headname.h

questa verrà immediatamente convertita in un file **h.headname**.

Sempre in fatto di caratteristiche generali (sintassi, metodi di immagazzinamento e modalità di compilazione) il run all'Acorn C-compiler è stabilito nella forma generica:

cc *filename -options,*

dove «filename» è il nome del file sorgente da cui si ricaverà il codice-oggetto e dove «options» rappresenta il tipo di selezione che il programmatore può fare sui quindici tipi di controllo che il compilatore ci consente di usare. Dal controllo **-link** che dopo la compilazione linka il file oggetto alle librerie standard; al controllo **-arthur** (dell'header <arthur.h>), con il quale, oltre a linkare alle suddette librerie, si opera lo stesso tipo di aggancio con le librerie di sistema (Arthur). L'opzione **-super** a sua volta è pronta per agganciare una libreria del sistema operativo Brazil e quella **-spool** a registrare l'output di schermo nel file **$.tmp.clog.**

Altri tipi di controlli interessanti da citare sono la **e***letter*, usabile per la soppressione degli errori marginali e la **f***letter* che inserisce, per mezzo di cinque, ulteriori sotto-opzioni, dei commenti al codice-sorgente in lavorazione.

Passando poi per le opzioni **k** e **K** che generano codici di controllo per il conteggio delle esecuzioni, concludiamo con la **p***letter digit*, con la quale è necessario portarci sulla direttiva di preprocessore **#pragma**, dedicato dall'Ansi alle istruzioni di compilazione.

Le direttive **#pragma** sono preposte al controllo del corretto comportamento del compilatore, legando a tale, propria

caratteristica, quella della particolare implementazione.

Le direttive **#pragma** esemplificate nel manuale sono di tre tipi: **-an**; **-bn**; **-cn**. La direttiva **#pragma -an**, offre la possibilità di abilitare (n=1) o disabilitare (n=0) messaggi di avvertimento circa l'uso di funzioni di cui non c'è un prototyping a cui riferirsi.

Anche la **#pragma -bn** fornisce dei «warning messages» (operazioni di cast fra puntatori ed integer), mentre la terza forma di direttiva, la **#pragma -cn**, abilita (o disabilita) la generazione di codici che controllano tutti gli accessi alla memoria.

Implementazione ed eccezioni

Dopo aver fatto qualche rapido accenno sulle modalità di run, controllo, compilazione e linkaggi vari, entriamo ora nel vivo dell'argomento affrontando le peculiarità dell'implementazione Acorn rispetto allo standard Ansi e le eccezioni allo stesso.

La prima grossa differenza che si incontra è quella legata alla categoria degli **identificatori**. Limitati com'erano all'ottavo carattere significativo nei «vecchi» compilatori pre-Ansi e da questa standardizzazione già obbligatoriamente aumentati a 31, nell'implementazione Acorn, gli identifier possono ora essere inseriti senza limiti di lunghezza. Ricordandoci di non utilizzarli in numero comunque inferiore a 31, sarà poi il compilatore stesso ad operare un suo «truncate» al 256esimo carattere significativo.

Tipo di File	Subdirectory
program source	c
header	h
object code	o
program	p

Figura 1

Per quanto poi riguarda i tipi base di dati (fig. 4), possiamo dire che il tipo **char** è classificato **unsigned** per default, ma che può essere liberamente dichiarato come **signed char** oppure **unsigned char** (ciò in forma ovviamente esplicita). Per l'elemento **float** poi, c'è da aggiungere che il tipo a quadrupla precisione **long double** è suscettibile di variazioni e che le quantità calcolate sono immagazzinate nel formato IEEE. La **long double**, così come la **double**, immagazzinano segno, esponente e parte più significativa della mantissa, al più basso indirizzo.

Al riguardo di tutto ciò che concerne le caratteristiche dei tipi aritmetici, il manuale offre tutta una serie di tabelle di riferimento sui vari range e digit a base 2 e 10, dei relativi float, double e long double. Dopo aver dato un'occhiata alla figura 5 relativa alle caratteristiche del floating point ed il raggio di azione dei suoi tipi, riprendiamo il lungo elenco delle particolarità della implementazione, con la trattazione dei tipi di dati strutturati.

Come si sa, lo standard Ansi lascia definire ad ogni tipo di implementazione, il layout dei componenti le proprie strutture. Su ciò il manuale ci rende una rapida schematizzazione delle regole imposte dall'Acorn C-compiler, specificando in una specie di decalogo i vari «comandamenti» ai quali obbedire. Tra questi, leggiamo ad esempio che i componenti del tipo **char** sono inseriti nei byte immediatamente adiacenti e che quelli del tipo **short** vengono allineati sugli stessi byte d'indirizzo; che il tipo **int** (l'unico valido per il bit field) può essere **signed** oppure **unsigned** e che un relativo bit field è comunque trattato, per default, come unsigned. Altre specificazioni vengono quindi fatte per i puntatori e, malgrado che il compilatore rispetti tutte le operazioni aritmetiche stabilite dall'Ansi, anche su integrali e floating.

A)

```
#include <stdio.h>
void main ()
{
      printf("Hello world!"\n"),
}
```

B) *fase di compilazione*:

```
cc hello
```

messaggi di compilazione:

```
Norcroft C 1.40 Jun 15 1987
Done: 0 warnings, 0errors, 0serious errors
```

C) *linkaggio:*

```
cc hello -link
```

D) *esecuzione:*

```
run p.hello
```

Figura 2

A) COMPILAZIONE MULTIPLA (e linkaggio)

cc mainprog util extra -link

(compila i file-sorgente c.mainprog, c.util, c.extra)

B) LINKAGGIO MULTIPLO SU CODICI-OGGETTO

link -file *filenames* **-library** *libnames* **-image** *progname* **-adfs**

libnames (-link) = $.arm,clib.o.ansilib

libnames (-arthur) =
$.arm. clib.o.ansilib,$.arm.clib.o.arthurlib

Esempio:

```
link -file o.mainprog,o.util,o.extra -library
 $arm.lib.o.ansilib -image p.mainprog -adfs
```

Figura 3