Università degli Studi di Bergamo
Facoltà di Ingegneria dell'Informazione

Informatica III

Elaborato 2: Cyclone

Revisione	Data	Redatto
1.1	2006/02/09	Silvio Moioli

Indice dei contenuti

Storia delle revisioni

Rev. n	Data	Redatto	Descrizione
1.0	2006/02/08	Silvio Moioli	Versione iniziale.
1.1	2006/02/09	Silvio Moioli	Aggiunti tutti i qualificatori.

Introduzione

Scopo del documento

Presentazione del secondo elaborato del corso, utilizzo del linguaggio Cyclone (dialetto del C sicuro) e del suo compilatore.

Definizioni, abbreviazioni e sigle

- ANSI, American National Standards Institute;
- ASCII, American Standard Code for Information Interchange;
- CVS, Concurrent Versioning System.

Riferimenti bibliografici e Web

- http://cyclone.thelanguage.org/, homepage del linguaggio Cyclone;
- http://en.wikipedia.org/wiki/Cyclone_programming_language, pagina Wikipedia di Cyclone;
- http://www.macports.org/, pagina del progetto MacPorts.

Approccio seguito in questo elaborato

L'elaborato consiste nell'adattamento di un programma, inizialmente scritto nel linguaggio C++, a Cyclone. L'adattamento è avvenuto in due fasi: una prima fase in cui il programma è stato adattato da C++ ad ANSI C, non presentata in questo documento, e una seconda fase in cui il codice C è stato adattato a Cyclone.

Si è preferito l'adattamento di codice esistente alla scrittura di un nuovo programma per meglio comprendere lo sforzo necessario a creare programmi C type-safe.

Presentazione del programma

Funzionalità principali

L'oggetto di questo elaborato è una libreria per la gestione di file contenenti collezioni di stringhe organizzate in hash table e un piccolo programma dimostrativo.

La libreria è composta da:

- un modulo per la gestione di file con record di lunghezza fissa che permette di aprire un file, leggere e scrivere record;
- un modulo per la gestione dei file come hash table, permette di aggiungere record, eseguire ricerche ed ottenere dati statistici come il numero di collisioni avvenute (utilizza le funzioni definite dal modulo precedente);
- un programma dimostrativo.

Funzioni principali

Qui sotto è riportato l'elenco delle funzioni del programma con una breve descrizione, diviso per file di dichiarazione.

- FileIO.h – libreria per la gestione di file di record a lunghezza fissa
  - newFile: apre e inizializza un file di record;
  - fileRead: legge un record da file;
  - fileWrite: scrive un record da file;
- HashFile.h – libreria per la gestione di file come hash table
  - newHashFile: apre e inizializza un file come hash table;
  - hashFileAdd: aggiunge un record;
  - hashFileWhere: ritorna la posizione di un record nel file;
  - getCollisions: ritorna il numero di collisioni avvenute;
  - getInsertions: ritorna il numero di record inseriti;
- main.c – programma dimostrativo
  - main: funzione principale.

Esempi delle modifiche apportate

Vengono qui di seguito riportati alcuni esempi delle modifiche apportate nell'adattamento da C a Cyclone. Dal punto di vista metodologico si è adottato un approccio incrementale iniziando dalle classi più a basso livello procedendo verso l'alto, man mano testando le funzioni ottenute.

Qualificatore `@notnull` (o puntatore `@`)

Spesso nei programmi C si fa l'assunzione che i puntatori utilizzati, particolarmente quelli ritornati da funzioni, non siano mai nulli. Questo può portare facilmente a errori (come la dereferenziazione del null), e Cyclone ha un qualificatore specifico per evitare che ciò accada. Infatti, in Cyclone è possibile marcare un puntatore come non nullo tramite l'apposito qualificatore `@notnull`, che fa sì che il valore venga controllato ad ogni assegnamento. Molte delle funzioni delle librerie base di Cyclone richiedono espressamente che i puntatori passati siano non-nulli. Seguono alcuni esempi di dichiarazioni di funzioni, parametri e variabili di questo tipo.

Da `FileIO.h`, dichiarazione di una funzione che ritorna un puntatore mai nullo:

```
/* Apre un file con il nome specificato in FILENAME e lo inizializza */
FILE* @notnull newFile();
```

L'implementazione della funzione, in `FileIO.cyc`, si occupa di terminare il programma e segnalare un errore in caso non fosse possibile ritornare un puntatore valido (ad esempio perchè non è possibile aprire il file in questione).

Da `HashFile.h`, dichiarazione di una funzione che riceve un puntatore `@notnull`:

```
/* Ritorna il numero di collisioni avvenute */
int getCollisions(HashFile* @notnull hf);
```

Da `HashFile.cyc`, dichiarazione di un puntatore `@notnull`:

```
/* Crea un nuovo file di hash */
HashFile* @notnull newHashFile(){
    HashFile* @notnull result = malloc(sizeof(HashFile));
...
```

Qualificatore `@numelts`

Tipicamente, nei programmi C, ogniqualvolta si opera con un vettore di dati è necessario conoscere anche la sua capienza massima; solitamente infatti le funzioni che accettano come parametro un vettore (o un puntatore) abbiano solitamente anche un parametro intero pari alla dimensione del vettore stesso. Possono sorgere errori, ad esempio, possono sorgere qualora venisse a mancare la coerenza fra la variabile che rappresenta la lunghezza del vettore e il valore reale della lunghezza stessa: in C non esiste un modo per assicurare che un puntatore punti a una locazione con un certo numero di posizioni utilizzabili.

Cyclone mette a disposizione un qualificatore per assicurare un certo numero di elementi in un vettore: `@numelts` (NUMber of ELemenTS).

Seguono alcuni esempi di dichiarazioni di funzioni, parametri e variabili di questo tipo.

Da `FileIO.h`, dichiarazione di una funzione che ritorna una stringa di lunghezza specificata:

```
/* Legge una stringa di lunghezza RECORDLENGTH dal file */
char* @notnull @numelts(RECORDLENGTH) fileRead(FILE* @notnull fp, int
pos, char* @notnull @numelts(RECORDLENGTH) s);
```

Da `FileIO.h`, dichiarazione di una funzione che riceve un puntatore `@numelts`:

```
/* Scrive una stringa di lunghezza RECORDLENGTH nel file */
void fileWrite(FILE* @notnull fp, char* @notnull @numelts(RECORDLENGTH) s, int
pos);
```

Da `HashFile.cyc`, dichiarazione di un puntatore `@numelts`:

```
/* Ritorna la posizione dove memorizzare un record nel file, -1 se fallisce */
```

```
int hashFileWhere(HashFile* @notnull hf, char* @notnull @numelts(RECORDLENGTH)
s){
char* @notnull @numelts(RECORDLENGTH) result = calloc(RECORDLENGTH,
sizeof(char));
...
```

Qualificatore `@fat` (o puntatore `?`)

Tra le caratteristiche essenziali che distinguono il C da altri linguaggi a più alto livello c'è la gestione esplicita della memoria e, quindi, l'aritmetica dei puntatori, che è presente anche in Cyclone.

Cyclone, però, permette l'aritmetica solo su un tipo particolare di puntatori, detti "fat", sui quali effettua controlli aggiuntivi per evitare accessi errati alla memoria come i buffer overflow. I puntatori "fat" possono essere dichiarati sia con il simbolo sintetico `?` che con il qualificatore `@fat`.

Un esempio di dichiarazione è il seguente (`main.c`):

```
int main(int argc, char* @fat * @fat argv){ ...
```

Da notare, fra l'altro, che tutti i puntatori a stringhe e vettori sono fat se non diversamente specificato, quindi

```
char* s = "pippo";
```

non è una dichiarazione valida in Cyclone, e restituisce un errore come il seguente:

```
test.cyc:4: s was declared with type char * but initialized with type const char
@{5U} (qualifiers don't match)
```

La dichiarazione va modificata come segue:

```
const char@ s = "pippo";
```

Qualificatore `@zeroterm`

La convenzione del C per il trattamento delle stringhe prevede che l'ultimo carattere sia sempre uno zero binario (codice ASCII zero). Nei programmi C frequentemente si trovano errori dovuti al mancato rispetto di questa convenzione: se una stringa non è terminata da 0, ad esempio, una chiamata a `strlen()` non ritorna il valore corretto.

Cyclone mette a disposizione uno strumento anche per questo caso: è infatti possibile dichiarare un puntatore con il qualificatore `@zeroterm` e le librerie assicureranno che il contenuto sia corretto a tempo d'esecuzione.

La dichiarazione è analoga ai casi precedenti, ad esempio:

```
const char* @fat @zeroterm s = "pippo";
```

Inizializzazione della memoria con `calloc()`

Nella maggior parte dei casi, Cyclone richiede che la memoria dinamica allocata sia inizializzata per ovvi motivi di sicurezza. La funzione `malloc()` della libreria standard del C, utilizzata nella maggioranza dei casi, non garantisce che la memoria allocata sia inizializzata, per questo in alcuni casi si è preferito usare la variante `calloc()` nell'adattamento a Cyclone; quest'ultima infatti azzera tutti i byte della memoria allocata. Questo è particolarmente utile per le stringhe e i puntatori `@zeroterm`. Esempio (tratto da `main.cyc`):

```
char* @fat format = calloc(10, sizeof(char));
```

Funzione strncpy()

La funzione `strncpy()` della libreria standard del C ha una semantica leggermente diversa dalla rispettiva versione Cyclone, è stato quindi necessario un piccolo adattamento.

In particolare, `strncpy()` copia al massimo n caratteri da una stringa a un'altra. Se la stringa sorgente è più corta di `len` caratteri, viene "allungata" con caratteri nulli (`'\0'`) fino a `len`, mentre se è più lunga la stringa di destinazione non viene terminata, come spiegato dal comando `man strncpy`, il cui output è riportato qui sotto:

```
Standard C Library (libc, -lc)


SYNOPSIS

#include <string.h>

char * strncpy(char * restrict dst, const char * restrict src, size_t
len);


DESCRIPTION

The strncpy() function copies at most len characters from src into dst.

If src is less than len characters long, the remainder of dst is filled

with '\0' characters.  Otherwise, dst is not terminated.
```

Segue da questa definzione che il programmatore C deve calcolare `len` tenendo conto del carattere `'\0'`, quindi, ad esempio, per copiare la stringa "pippo" è necessario porre `len` a 6. Nell'implementazione di Cyclone, invece, si suppone che sia la libreria ad assicurarsi della corretta terminazione della stringa, quindi il programmatore deve porre len al numero di caratteri utili escludendo il `'\0'` (nell'esempio precedente, 5). In caso contrario si ottiene un errore durante l'esecuzione; si legge infatti dal codice sorgente di Cyclone:

```
// Destructively copy at most n characters from src into dest
mbuffer_t<`r> strncpy(mbuffer_t<`r> dest, buffer_t src, size_t n) {
  int i;
  assert(n <= numelts(src) && n <= numelts(dest));
...
```

Esiste anche una variante che si avvicina maggiormente allo standard, `zstrncpy()`:

```
// Strncpy that does not pay attention to zero chars
mbuffer_t<`r> zstrncpy(mbuffer_t<`r> dest, buffer_t src, size_t n) {
  assert(n <= numelts(dest) && n <= numelts(src));
...
```

Aspetti implementativi

Installazione di Cyclone su Mac OS X

Per installare Cyclone su Mac OS X si può usare il sistema MacPorts. Una volta installato, con il comando:

```
sudo port install cyclone
```

si può installare la versione più recente di Cyclone, che viene automaticamente scaricata dal CVS e compilata insieme alle relative dipendenze. A causa di un difetto di MacPorts, però, una cartella non viene installata correttamente; è necessario quindi eseguire anche il comando:

```
sudo ln -s /opt/local/lib/cyclone /opt/local/lib/cyclone
```

per ottenere un'installazione funzionante.