Laboratorio di sistemi operativi
A.A. 2010/2011
Gruppo 2
Gennaro Oliva
14
Breve riepilogo sul linguaggio C

Seminari

- Studenti volontari e volenterosi sono invitati a contattare docente per preparare seminari di 15 minuti sui seguenti argomenti

1) GDB debugger del progetto GNU

2) Make utility per l'automatizzazione della compilazione del codice

3) ctags/cscope utility per l'esplorazione del codice sorgente

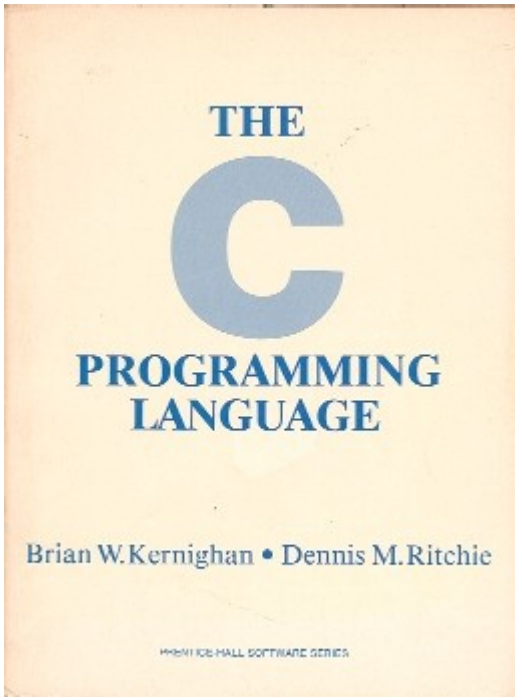
4) Valgrind libreria per il debug di problemi di memoria

Breve storia: gli albori

- Lo sviluppo del C ha inizio nel 1969 nei Bell Labs di AT&T
- Il nome C fu scelto per evidenziare la sua discendenza dal linguaggio B progettato da Ken Thompson
- La prima versione fu realizzata per agevolare il porting di Unix dal PDP-7 al PDP-11
- Nel 1973 il kernel fu riscritto interamente in C, rendendo Unix uno dei primi sistemi operativi implementati in linguaggio diverso dall'assembler

Breve storia: K&R C

- Nel 1978, Brian Kernighan e Dennis Ritchie pubblicarono la prima edizione di “The C Programming Language”
- Il libro è stato per anni la specifica ufficiosa per il linguaggio di programmazione
- La versione del linguaggio descritta in questo libro di testo viene comunemente detta K&R C

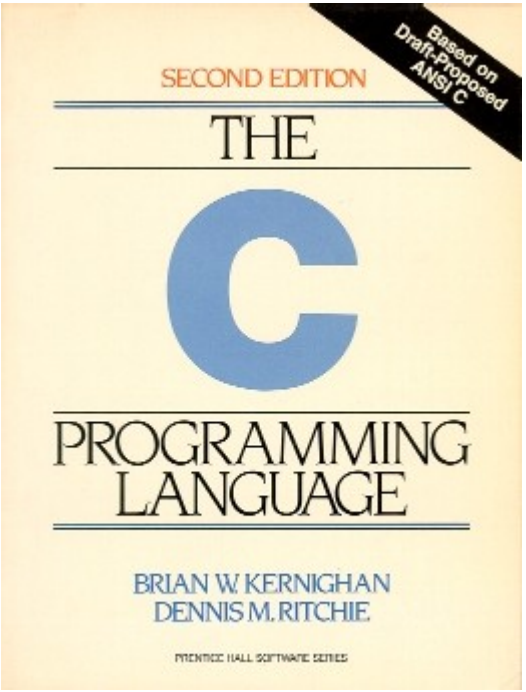


Breve storia: ANSI C

- Durante gli anni 70 e 80 le versioni di C proliferarono
- Nel 1983 l'American National Standards Institute (ANSI) diede vita ad una commissione per stabilire una specifica standard del linguaggio che includesse le nuove caratteristiche presenti nei vadi dialetti del K&R: i prototipi di funzione, i puntatori void, il supporto a caratteri internazionali e la localizzazione
- Nell'89 venne ratificato lo standard ANSI C chiamato anche C89 che nel 90 fu adottato dall'International Organization for Standardization (ISO)

Breve storia: ANSI C

- Lo standard C89 è supportato da diversi compilatori C ed è la versione di riferimento con cui è scritta la maggior parte del software esistente
- Le specifiche del C89 sono descritte nella seconda edizione del libro di Kernighan e Ritchie



Breve storia: C99

- Lo standard fu modificato nel 1995 e successivamente nel 1999 anno in cui fu pubblicata la versione C99
- C99 introdusse diverse caratteristiche nuove già implementate in diversi compilatori, quali le funzioni inline, nuovi tipi di dato tra cui long long int e complex, array di lunghezza variabile, macro con numero di argomenti variabile, supporto per i commenti con //

Breve storia: C1X

- C1X è il nome non ufficiale del nuovo standard C in via di progettazione che rimpiazzerà il C99 includendo le nuove funzionalità disponibili nelle recenti implementazioni dei compilatori
- Lo standard ancora in fase di definizione, supporterà tra l'altro il multithreading, la funzione gets_s come alternativa sicura a gets per evitare buffer overrun, la modalità create-and-open per fopen per il locking

Caratteristiche del C

- Il C è un linguaggio di programmazione essenziale che utilizza una sintassi semplicemente traducibile in linguaggio macchina
- Storicamente legato a UNIX ma utilizzato per la programmazione di diversi sistemi operativi e di sistemi embedded per la portabilità e l'efficienza
- Linguaggio general purpose utilizzabile per lo sviluppo di applicazioni numeriche, database, ...

Traduzione del codice

- Il linguaggio C è un linguaggio compilato ovvero il codice di un programma prima di poter eseguito viene tradotto in linguaggio macchina, secondo un processo che avviene in tre fasi:
- Fase 1: Preprocessing
- Fase 2: Compilazione
- Fase 3: Linking

Preprocessing

- Nella fase di preprocessing il sorgente viene modificato secondo delle direttive specificate dal programmatore
- Le direttive al preprocessore sono contenute nelle righe che iniziano con il carattere “#”
- Si distinguono tre classi di direttive:
  - #include per l'inclusione di file
  - #define per la definizione di costanti e macro
  - #if, #ifdef, #ifndef, #else, #elif and #endif per le direttive condizionate

La direttiva include

- La direttiva include indica al preprocessore di sostituire la riga con il contenuto di un header file
- `#include <headerfile.h>`
  cerca headerfile.h nelle directory di sistema
- `#include “myheader.h”`
  cerca nelle directory che contiene il file sorgente
- Gli header file contengono:
  - direttive per il pre-processore
  - prototipi delle funzioni
  - dichiarazioni di variabili e strutture
- Gli header file non devono contenere l'implementazione delle funzioni

La direttiva define

- La direttiva #define utilizza la sintassi:

```
#define <identificativo> <rimpiazzo>
#define <identificativo>(<parametri>) <rimpiazzo>
```

- Viene utilizzata per definire costanti

```
#define PI 3.14159
```

oppure macro

```
#define RADTODEG(x) ((x) * 57.29578)
```

- Dopo la fase di preprocessing nel codice vengono sostituite tutte le occorrenze degli identificativi con il rimpiazzo

Le direttive condizionali

- Consentono di introdurre delle modifiche al programma in fase di traduzione in base a parametri specificati
- Per utilizzare l'header file giusto in base al sistema operativo su cui si compila il codice:

```
#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif
```

- Per abilitare stampe di messaggi di debug nelle versioni del di sviluppo del codice

```
#if DEBUG_LEVEL > 0
 fprintf(stderr, … );
#endif
```

Compilazione

- La compilazione è il processo di traduzione del codice sorgente generato nella fase di preprocessing in file oggetto
- I file oggetto sono file binari contenenti istruzioni in linguaggio macchina corrispondenti al codice C che abbiamo scritto
- I file oggetto sono rilocabili ovvero gli indirizzi di memoria dei simboli (funzioni, strutture dati,...) in essi contenuti partono da `0`
- La compilazione non genera programmi eseguibili

Compilazione

- Durante la compilazione viene verificata la sintassi del programma e la compatibilità tra i tipi di dato utilizzati
- Molti compilatori suddividono questa fase in due parti
  - Traduzione dal C all'assembler
  - Traduzione dall'assembler al codice oggetto

Linking

- Nel processo di linking i file oggetto vengono combinati in un unico il file eseguibile
- Durante questo processo tutti i simboli contenuti nei file oggetto vengono sistemati in un unico spazio di indirizzamento eseguendo la rilocazione gli indirizzi
- In questa fase il linker controlla l'esistenza di tutte le funzioni invocate nel programma e delle variabili utilizzate, cosa che non avviene durante la compilazione
- Il compilatore compila un file per volta pertanto quando non trova una funzione menzionata nel codice assume semplicemente che questa è definita in un altro file
- Il linker invece nella produzione dell'eseguibile deve verificare necessariamente la presenza di tutte le funzioni ed eventualmente segnalarne l'assenza

Librerie

- I linker possono utilizzare oggetti contenuti in collezioni chiamate librerie
- Le librerie sono collezioni di funzioni pronte per l'uso, che risolvono classi di problemi
  - Librerie standard
  - Librerie matematiche
  - Librerie per immagini
  - Librerie multimediali
  - ...

Linking dinamico

- I sistemi operativi moderni consentono il linking dinamico ovvero di posporre la risoluzione di alcuni simboli durante l'esecuzione del programma
- In tal caso il file eseguibile contiene dei simboli non risolti che vengono collegati agli effettivi oggetti nell'esecuzione dal loader
- Questa tecnologia fornisce due vantaggi:
  - Le librerie d'uso comune possono essere memorizzate in locazioni di memoria condivise e non devono necessariamente essere duplicate per ogni singolo programma
  - Quando modifichiamo una funzione (per eliminare un bug ad esempio) non dobbiamo ricompilare tutti gli eseguibili che la utilizzano

Perché separare compilazione e link

- Semplificano la manutenzione del codice: nella compilazione di programmi grandi, quando si modificano determinate funzioni nel creare l'eseguibile è sufficiente compilare i file contenenti tali funzioni ed effettuare il linking
- Il comando make è uno strumento prezioso
- Favorisce lo sviluppo di librerie ed il riuso del software
- Conoscere le tre fasi della traduzione del codice aiuta nella di bug

Il compilatore gcc

- GNU Compiler Collection (GCC) è un insieme di compilatori sviluppato dal progetto GNU
- GCC viene utilizzato dalla maggior parte dei sistemi operativi Unix moderni (Linux, BSD, Mac OS X, ...), supporta una vasta gamma di architetture e viene utilizzato per la maggior parte di sistemi embedded
- Il nome originale era GCC (GNU C Compiler) poiché supportava solo il linguaggio C ma fu cambiato quando fu introdotto il supporto ad altri linguaggi tra cui: C++, Fortran, Pascal, Objective-C, Java e Ada

gcc: istruzioni per la traduzione

- Dopo aver scritto il codice del programma con il nostro editor preferito (vi, emacs, gedit, ...) per compilarlo possiamo utilizzare l'istruzione

  $ gcc programma.c

- La singola istruzione effettua tutte e tre le fasi della traduzione invocando il preprocessore (cpp) ed il linker (ld) in modo trasparente
- In caso di assenza di errori nel programma il comando genera un file eseguibile chiamato a.out che ha già i permessi di esecuzione
- Per dare un nome differente utilizziamo l'opzione -o

  Esempio:

- $ gcc -o myprog programm.c

  per nominare l'eseguibile myprog.

Opzioni

- Il comando gcc accetta 3 set di opzioni separati per le 3 fasi della traduzione
- Opzioni per il preprocessore come ad esempio -Dmacro[=defn] che di permette di effettuare una #define in fase di compilazione
- Opzioni per il compilatore come ad esempio le opzioni di ottimizzazione -O1 -O2 -O3
- Opzioni per il linker come ad esempio -l library che segnala al linker il nome della libreria che contiene le funzioni utilizzate nel nostro programma

gcc: istruzioni per la compilazione

- Per comunicare a gcc di effettuare soltanto la fase di compilazione senza il linking si utilizza l'opzione -c
- Supponendo che il nostro programma sia strutturato in diversi file sorgente: main.c io.c processing.c
- Le istruzioni:
  gcc -c main.c
  gcc -c io.c
  gcc -c processing.c
- generano i file main.o io.o e processing.o mentre l'istruzione:

  gcc -o myprogram main.o io.o processing.o
- genera il programma eseguibile myprog

La funzione main

- In ogni applicazione esiste sempre un'unica funzione main unica per l'applicazione non “unica in ogni file”
- Il prototipo standard della funzione main è il seguente:

  int main(int argc, char *argv[])
- Gli argomenti della funzione main consente di ottenere i parametri passati al programma sulla linea di comando in fase di esecuzione
- argc è un intero contenente il numero di argomenti che si trovano sulla riga di comando (vale almeno 1)
- argv è un array di puntatory ad array di carattere contenenti le stringhe passati come argomenti
- argv[0] punta al primo argomento (il nome del programma)
- argv[1] punta al secondo argomento
- ...

Prototipi

- Prima di utilizzare ogni funzione all'interno di un codice è necessario dichiararne il prototipo
- Per le funzioni di libreria e sufficiente inserire l'header file corrispondente
- Per le nostre funzioni è opportuno specificare i prototipi all'inizio del file
- Se le nostre funzioni vengono utilizzate da più file è opportuno scrivere un header file con i prototipi in modo da poterlo includere in tutti i file che le utilizzano
- Il prototipo specifica:
  - il tipo del valore restituito dalla funzione
  - il nome della funzione
  - l'elenco dei parametri con rispettivi tipi

```
int average (int *);
```