

# Alcuni errori tipici

Frammento 1

# Sulla gestione errori...

- Deve essere controllato l'esito delle funzioni di libreria
  - malloc-calloc non controllate!
- Mai fidarsi del valore dei parametri passati a una f.ne di libreria
  - testare sempre se i valori sono consistenti con quelli che ci aspettiamo (if, assert etc...)
- Le funzioni che modificano errno devono documentare la cosa
  - modello man, basta nel commento all'inizio

# Sulla gestione errori... (2)

- Erro viene interpretata da perror
  - è necessario assegnare a errno solo valori noti (ENOMEM, EINTR etc..)
- Le stampe di messaggi di errori devono essere effettuate su stderr
  - perror(...) != fprintf(stderr, ....)
- è preferibile non fare stampe di errori o di debug dentro le funzioni di libreria ma ritornare un valore (settando errno opportunamente)
  - come accade per le librerie che usiamo di solito

# Sulla gestione errori... (3)

- è VIETATISSIMO chiamare la `exit()` o la `_exit()` da dentro le funzioni di libreria

# Memoria

- Se si effettuano più allocazioni in sequenza
  - è necessario liberare tutta la memoria precedentemente allocata se una allocazione va male
- è preferibile effettuare malloc separate per diverse variabili e free separate delle stesse
  - accedere a indirizzi dinamici non generati da malloc direttamente può generare problemi di portabilità e complica la leggibilità del programma
- dove possibile è sempre meglio usare variabili locali piuttosto che dinamiche
  - più efficiente, non ‘genera’ memory leak

# Commenti

- Non devono ripetere passo passo quello che è evidente dal codice!
- è preferibile:
  - scrivere un commento esauriente all'inizio della funzione, con eventuali note algoritmiche
  - commentare sinteticamente all'inizio di un blocco di 5-15 statement per spiegare cosa sta per succedere
  - commentare bene le variabili globali (inclusa la politica di modifica)
  - commentare brevemente le var locali dal significato non ovvio

# Miscellanea

- Le funzioni private (che servono solo in un file) devono essere dichiarate *static*
- è *PESSIMA norma* ridefinire i nomi delle keyword C in modo che il programma sembri più Java-like
  - questo complica la vita a chi legge ed a chi scrive
- Non si devono usare caratteristiche non standard ISO
  - mixed decl and code, array dinamici, funzioni annidate

# Miscellanea

- Leggere sempre bene il man
  - e non assumere niente che non sia scritto lì
  - es: `memcpy(NULL, ...)` segfault su molti sistemi
- Molte `remove` sono lunghe e contorte
  - in questi casi c'è qualcosa che non va
- Funzioni grosse e contorte o troppo piccole e frammentate rendono il codice difficile da leggere e da mantenere
- la ricorsione per molti è ancora una bestia strana (alcuni usano `var` di appoggio inutili ...)



# Miscellanea

- Meglio `strtod()` che `atof()` nelle conversioni
  - Se no non si riesce a gestire gli errori
- Molti di coloro che hanno scritto i commenti `doxygen` non si sono preoccupati di compilare ed andare a gestire i warning!
- Nel `make` spesso sono state dimenticato le dipendenze da `include` annidati (usate **`gcc -MM !!!`**)

# Miscellanea

- È VIETATISSIMO includere i .c in altri .c
  - Questo crea duplicazione di codice nei .o e contravviene a tutte le norme di organizzazione del codice C
- È anche da evitare l'inclusione di .h in altri .h
  - Questo permette di evitare dipendenze non necessarie e nascoste in innocui include
- Nell'accesso degli elementi di un array è sempre da preferire la notazione `a[i]` alla `*(a+i)`
  - Sono equivalenti ma la prima rende il codice più leggibile e manutenibile