Riassunto comandi latex

Matteo Mistri

12 ottobre 2017

Indice

Capitolo 1

Introduzione

Linguaggio di basso livello, poco sopra linguaggio macchina. Pochissime funzionalità di base, per funzioni più avanzate devo importare librerie esterne, ma esco dallo standard e non è detto funzionino su tutti i sistemi. Manca il tipo string, sostituito dal puntratore a puntatori di char.
Si cerca di dare lo scopo minimo alle variabili, dichiarandole nel blocco più interno possibile. Le variabili statiche hanno come scopo il file intero, quindi rimangono in memoria anche dopo il termine di chiamata della funzione. Non può essere ricreata, se l'esecuzione ripassa da variabili statiche, esse vengono saltate.
Ci sono tipi specifici che sostituiscono int per forzare una dimensione costante, indipendente dall'architettura(Es: `uint_32` rappresenta un intero senza segno lungo 32 bit).

Capitolo 2

Linguaggio

2.1 Main

argc contiene il numero di argomenti che riceve, argv è un array di array di char(quindi un'array di stringhe) che contiene l'input a partire dall'indice 1.

ţnt main(int argc, char **argv) int input = atoi(argv[1]);

Devo usare una funzione di conversione, contenuto nella stdlib, per convertire l'alfanumerico in intero atoi(argomento).
Per usare la printf devo usare i segnaposto con i

2.2 Array

Posso dichiarare un'array come:

ţipo nomearray[dimensione];

dimensione può essere una variabile int. Dichiararlo in questo modo permette di gestire la memoria in modo automatico, senza gestione esplicita.
Se non so a priori la dimensione come faccio? Mi serve una dimensione dell'array "dinamica", non posso allocare l'array a priori. Posso aggiungere elementi a una lista e, una volta completata, li copio in un'array in modo da sapere la dimensione precisa.
Le liste non fanno parte della libreria standard, ma di glib- Se alloco memoria

per una variabile che devo ritornare, allora la free andrà scritta dopo la chiamata della funzione.

2.3 GGO

Se creo un file .ggo associato ad un programma, esso mi crea l'help dei parametri del programma. Ha una sua sintassi specifica. Un programma apposito permetterà di generare il pezzo di programma c che esegue il parsing delle opzioni da riga di comando. Ha la funzione di parser per gli argommemnti passati da linea di comando, in modo che siano accessibili dal programma in modo agile. Il makefile può automatizzare questa compilazione. Il programma genera i file commandline.g e commandline.h e io non devo effettuare alcuna modifica. Nel programma dovrò solo includere "cmdline.h" di modo da generare dipendenza tra il mio programma e il cmdline.

ştatic struct `gengetopt_args_info args_info`;
assert(`cmdline_parser`(argc, argv, `&args_info`) == 0);

// Le istruzioni creano una struct che contiene i dati parsati. Per accedervi utilizzo `args_info.parametrodaestrarre_args`

2.4 Varie

assert verifica una condizione. Se non è soddisfatta fa un return 0

calloc(n, sizeof(tipo)) è una malloc che inizializza n caselle di memoria di dimensione tipo, iniziandole tutte a 0

char* copia = strndupa(puntatoreastringa, m) prende la stringa passata e ne estrapola i primi m caratteri. L'allocazione della memoria necessaria è dinamica, non ho bisogno di malloc e free(viene fatta automaticamente alla fine della funzione chiamante)

2.5 Funzioni

Sono definite come nel C++
Gli argomenti passati a una funzione non vengono modificati nello spazio di chi chiama, ma solo della funzione. Se voglio che le modifiche siano "definitive", nella chiamata della funzione passo l'indirizzo della variabile e nella funzione uso come argomento un puntatore.

Capitolo 3

Valgrind

Valgring esegue un programma tracciando l'uso della memoria, mostrando un report della memoria in uso, quella allocata, quella liberata e quella non allocata