LINGUAGGIO C

Introduzione

Qual è la funzione di un compilatore?
Un compilatore è un programma che traduce il codice sorgente in codice oggetto, ovvero traduce le informazioni scritte in un linguaggio di programmazione nel linguaggio macchina del computer.
Qual è la funzione di un interprete?
Un interprete è un programma che traduce il codice sorgente, scritto in un linguaggio di programmazione ad alto livello, in istruzioni eseguibili dal computer, che vengono eseguite direttamente. Questo processo avviene in tempo reale durante l'esecuzione del programma.
Cosa è un linguaggio assemblativo (assembler)?
Un linguaggio assemblativo, o semplicemente assembler, è un linguaggio di programmazione di basso livello utilizzato per scrivere codice macchina per un determinato processore o architettura hardware. Consente di scrivere istruzioni direttamente comprensibili dal processore, senza la necessità di ulteriori traduzioni.
Perché nasce il linguaggio C?
Il linguaggio C è stato progettato per semplificare la programmazione di sistemi operativi, consentendo agli sviluppatori di scrivere codice di sistema che fosse altamente portabile e potente. C offre ampio controllo sui dettagli dell'hardware e delle operazioni di basso livello, permettendo di ottimizzare il codice per massimizzare le prestazioni.
Quali sono i principi fondamentali del linguaggio C?
Alcuni prin_C_ples:
- è conciso, le parole chiave e i simboli degli operatori sono brevi e si evita ogni ridondanza
- delega tutto il possibile alla libreria standard C
- il codice generato può non aver bisogno di software di supporto (ambiente di esecuzione)
- non ci sono controlli automatici a runtime
- un programma è sintatticamente una collezione di funzioni
- il programma principale è la funzione di nome main

In che linguaggio è scritto il Compilatore C? Perché?
Il compilatore C attuale è stato prodotto grazie al bootstrapping.
Il primo compilatore C fu scritto in Assembly. Questo compilatore fu poi utilizzato per scrivere un compilatore C completo ma poco efficiente il quale fu poi utilizzato per scrivere un compilatore C completo ed efficiente.
Cosa è un cross-compiler?
È un tipo di compilatore che genera codice oggetto per una piattaforma diversa da quella sulla quale il compilatore stesso è in esecuzione.
Cosa si intende per bootstrapping?
È una tecnica usata per produrre un compilatore scritto nel linguaggio di programmazione che intende compilare.

Concetti base

Il comando gcc: cosa fa? Quali sono i passi gestiti da gcc?
Il comando gcc prende un file sorgente e tramite vari passi permette di ottenere un file oggetto eseguibile.
I passi gestiti da gcc sono:
- preprocessore
- compilatore
- ottimizzatore
- assemblatore
- linker

Cosa sono i file con suffisso ".c"? E i file .h?
I `.c` sono file sorgenti, i quali contengono il codice effettivo del programma scritto in C.
I `.h` sono file di intestazione, i quali contengono dichiarazioni e informazioni sull'interfaccia del codice.

Come si selezionano solo alcuni dei passi di gcc?
Per generare solo il codice preprocessato si usa `-E`.
Per generare il file assembler (compilato ed ottimizzato) si usa `-S`.
Per generare solo il file oggetto si usa `-c`.

Cosa è in grado di fare il linguaggio C senza la libreria standard?
L'unico comportamento garantito per un programma riguarda i valori di oggetti "volatile".

In realtà in linux l'esecuzione del programma inizia dalla funzione _start, non main: perché?
La funzione `_start` è il punto di ingresso del programma direttamente gestito dal sistema operativo, mentre `main` è la funzione del programma scritta dall'utente.
Si occupa di:
- inizializzare il programma,
- gestire i parametri dalla riga di comando,
- chiamare la funzione main,
- gestire la terminazione del programma.

Come fa un programma C a eseguire una system call?
Può utilizzare le funzioni di libreria standard per eseguire le system call.

Come fa un programma C senza libreria standard a eseguire una system call?
Eseguire una system call senza utilizzare la libreria standard richiede l'uso di chiamate di sistema dirette, spesso utilizzando assembly inline o accesso diretto alle interfacce di sistema del sistema operativo.

Come si fa in C a leggere o scrivere dati a un indirizzo costante della memoria (o del bus)?
Si può accedere direttamente ad un indirizzo costante di memoria o bus utilizzando un puntatore.

Come si è evoluto il linguaggio C nel tempo e quali sono gli standard?
C fu standardizzato da ANSI nel 1989 e successivamente ISO/IEC a fine millennio. Lo standard C più recente è ISO/IEC 9899:2018 del 2018. Ogni nuovo standard ha introdotto nuove funzionalità, rimanendo comunque retrocompatibile.

Come si costruisce il programma eseguibile per un programma costituito da molteplici file sorgenti?
Bisogna compilare ciascun file sorgente in un file oggetto e poi linkare tutti i file oggetto per creare il programma eseguibile finale. Se il programma include librerie esterne, bisogna linkare anche queste.

Cosa sono le librerie?
Sono raccolte di codice (funzioni, procedure, classi o moduli) scritte da sviluppatori e messe a

disposizione per essere riutilizzate da altri programmatori.

Cosa significa linking statico e linking dinamico?

`linking statico` : le librerie necessarie vengono incorporate direttamente nel file eseguibile durante la fase di compilazione.

`linking dinamico` : il file eseguibile fa riferimento alle librerie esterne senza incorporarle direttamente. Il collegamento ad esse avviene a runtime.

Linguaggio

Caratteri ammessi, commenti, spazi, indentazione

Caratteri ammessi: lettere (maiuscole e minuscole), cifre, simboli speciali e caratteri di punteggiatura. C è case sensitive.

Commenti di due tipi: i commenti singola riga iniziano con il simbolo `//` e continuano fino alla fine della riga, i commenti multilinea iniziano con `/*` e terminano con `*/`.

Gli spazi bianchi non influenzano il comportamento del programma, sono utilizzati per migliorare la leggibilità del codice.

Si può indentare il codice all'interno dei blocchi per mostrare la struttura del programma in modo chiaro.

Identificatori leciti

Un identificatore deve iniziare per underscore o una lettera (maiuscola o minuscola). Può contenere lettere, cifre o underscore e non può essere una keyword riservata.

Definizione di variabili

Una variabile si dichiara con la sintassi `tipo nome_var;`.

Si può inizializzare al momento della dichiarazione con `tipo nome_var = valore;`.

Operatori aritmetici? qual è la loro funzione?

Simboli utilizzati per eseguire operazioni matematiche.

- `+` : addizione fra due operandi
- `-` : sottrazione fra due operandi
- `*` : moltiplicazione fra due operandi
- `/` : divisione fra due operandi (se entrambi int è divisione intera)

Interi con e senza segno

Un int con segno permette di rappresentare valori positivi e negativi, ~[-2B, +2B]. Un int senza segno rappresenta solo valori >= 0, ~[0, 4B], e si definisce con `unsigned`.

Interi di varia ampiezza

Diverse possibilità per memorizzare interi in base al numero di bit necessari. `char` (1 byte), `short` (2 byte), `int` (4 byte), `long` (4 o 8 byte), `long long` (8 byte).

Overflow

Un overflow si verifica quando il risultato di un'operazione supera il valore massimo rappresentabile.

Operatore modulo

`%` : calcola il resto della divisione intera fra due operandi.

Operatori di confronto

Confrontano due valori e restituiscono un risultato booleano.

- `>` : maggiore di
- `>=` : maggiore o uguale a
- `<` : minore di
- `<=` : minore o uguale a

- `==` : uguale a
- `!=` : diverso da

Operatori di shift

Permettono di spostare i bit di un dato di un numero specificato di posizioni.

- `<<` , shift a sinistra, sposta i bit verso sinistra
- `>>` , shift a destra, sposta i bit verso destra

Operatori bit a bit

Consentono di fare operazioni sui singoli bit.

- `&` fa AND tra due operandi
- `|` fa OR tra due operandi
- `~` fa NOT di un operando
- `^` fa XOR tra due operandi

Operatori di assegnamento

Assegnamento semplice con `=` , assegna il valore dell'espressione a destra all'operando a sinistra.

Operatori di assegnamento-modifica

Assegnamento con operazione, esegue prima l'operazione e poi assegna. I possibili operatori sono `+=` , `-=` , `*=` , `/=` , `%=` .

`x += 1` equivale a `x = x + 1`

L-valori/R-valori

Un L-valore è un'espressione che rappresenta un'area di memoria identificabile e modificabile.

Un R-valore è un'espressione che rappresenta un valore memorizzato in memoria.

Operatori di pre/post incremento/decremento

Incrementano o decrementano il valore di un'unità.

- Post incremento: `x++`
- Post decremento: `x--`
- Pre incremento: `++x`
- Pre decremento: `--x`

Operatori post vengono eseguiti per ultimi, operatori pre hanno effetto immediatamente.

Espressioni di controllo dell'esecuzione (quelle negli if/while ...)

Nelle espressioni di controllo è possibile utilizzare operatori di confronto e/o gli operatori logici.

Operatori logici (&& || ...) cortocircuitazione

Servono per eseguire operazioni logiche su valori booleani o condizioni. Essi sono:

- `&&` : AND logico, true se entrambe le espressioni sono true, false altrimenti.
- `||` : OR logico, true se almeno un'espressione è true, false altrimenti.
- `!` : NOT logico, true se l'espressione seguente è false, false altrimenti.

La cortocircuitazione avviene quando si interrompe la valutazione di un'espressione logica non appena il risultato finale è determinato. Nell'AND se il primo operando è false, nell'OR se il primo operando è true.

Espressione condizionale ?:

Operatore ternario associabile ad un if-inline.

```
condizione ? ramo_then : ramo_else
```

Istruzioni e blocchi {}

Quale è la sintassi e il significato delle istruzioni

Un'istruzione rappresenta una serie di comportamenti che deve mettere in atto il programma. Il punto

e virgola `;` è il terminatore, trasforma espressioni in istruzioni.

if, while, do...while, for

Costrutti per gestire il flusso del programma:

- `if (condizione) { ramo_then } else if (condizione) { ramo_then } else { ramo_else }`
- `while (condizione) { corpo }`
- `do { corpo } while (condizione)`
- `for (init; condizione; incremento) { corpo }`

Se il contenuto delle parentesi graffe è di una sola riga, le parentesi non sono necessarie.

`for (a; b; c) S` significa `a; while (b) {S; c;}`

break/continue

Istruzioni usate all'interno dei cicli per modificarne il flusso di esecuzione.

- `break` termina l'esecuzione del ciclo corrente.
- `continue` termina l'iterazione corrente e passa alla successiva (senza terminare il ciclo).

switch

Costrutto per effettuare una scelta basata sul valore di un'espressione. Utilizzabile solo con interi o char.

`switch (espressione) { case label: ... break; }`

switch: range di valori, default

Usando l'operatore `...` è possibile specificare un range di valori per la clausola del case. `case 1 ... 3:`.

Il case `default:` è inserito per ultimo e viene eseguito se nessun'altro case prima è stato scelto.

Tipi base

Integer: char, short, int, long, long long

Diverse ampiezze per memorizzare valori interi.

- `char` : 1 byte, [-128, 127]
- `short` : 2 byte, ~ [-32k, 32k]
- `int` : 4 byte, ~ [-2B, 2B]
- `long` : 4 o 8 byte, intervallo molto ampio
- `long long` : solitamente 8 byte, intervallo ampissimo

signed/unsigned

`signed` è un valore con segno (default), `unsigned` rappresenta solo i valori positivi (intervallo traslato).

Significato di char

Char è un intero ad 8 bit che viene usato per rappresentare caratteri ASCII.

Conversione fra tipi interi

intero -> intero più ampio: estensione con bit più significativi a zero.

intero -> intero meno ampio: si mantiene solo la parte meno significativa.

Floating point: float, double, long double

Rappresentano valori in virgola mobile.

- `float` : 4 byte, circa 6-9 cifre decimali. Es `float pi = 3.14f`
- `double` : 8 byte, circa 15-17 cifre decimali. Es `double pi = 3.14`
- `long double` : 10 o 16 byte, maggiore precisione di double. Es `long double pi = 3.14L`

La dimensione e la precisione effettive dei tipi possono variare a seconda del compilatore e

dell'architettura del sistema.

Conversione fra tipi floating point

float -> float: adattamento della mantissa con arrotondamento delle cifre decimali.

Conversione fra int/floating point

int -> float: conversione con cifre decimali a zero. 5 -> 5.0f

float -> int: troncamento della parte decimale. 5.7f -> 5

Nuovi tipi bool/_Complex

`bool` rappresenta i valori booleani true e false, introdotto nell'header <stdbool.h>.

`_Complex` rappresenta numeri complessi, cioè numeri con una parte reale e una parte immaginaria.

Il tipo void

Un tipo speciale utilizzato per rappresentare l'assenza di tipo. Ha diversi casi d'uso:

- come tipo di una funzione, per indicare che non ritorna alcun valore
- negli argomenti di una funzione, significa nessun parametro
- per fare casting esplicito su variabili, indica che una variabile non viene più usata dal programma
- per puntatori void, ovvero generici.

Costanti

Costanti numeriche intere e floating point

Sono valori inseriti direttamente nel codice senza essere dichiarati come variabili.

- decimali: classico valore intero o float. `42` o `3.14f`
- esadecimali: *0x* seguito da cifre esadecimali. `0x4d6`
- ottali: *0* seguito da cifre ottali. `0537`
- esponenziali: notazione scientifica *mantissa*e*esponente*. `2.5e3` = 2500.0

Costanti carattere

Valori letterali usati direttamente nel codice per rappresentare caratteri.

- ASCII: classici caratteri tra apici singoli. `'a'`
- escape: caratteri speciali backslash + carattere. `'\n'`
- interi: interi da 0 a 255 rappresentano caratteri ASCII. `65` = 'A'

Costanti stringa

Sequenze di caratteri racchiuse tra doppi apici utilizzate per rappresentare testo nel codice. Possono contenere sequenze di escape. `"hello world"`

Puntatori

Puntatori

Sono variabili che contengono l'indirizzo di una locazione di memoria.

`type* var_name`

Operatori & e *

- `&` : operatore unario di indirizzo, restituisce l'indirizzo della locazione di memoria dell'operando.
- `*` : operatore unario di dereferenziazione, restituisce il valore della variabile puntata dall'operando.

Il puntatore NULL

Indica un puntatore non valido. Viene memorizzato come 8 bit a zero.

Significato e uso di void *

Un puntatore generico che non fa riferimento ad un tipo di dato specifico. Non si può dereferenziare se non viene fatto un casting esplicito.

Aritmetica dei puntatori
puntatore + o - int: permette di accedere agli elementi a destra o sinistra dell'elemento puntato (ritorna un puntatore)
puntatore - puntatore: indica quanti elementi ci sono tra 2 puntatori (ritorna un intero)
Incremento/decremento di puntatori
Sposta il puntatore all'elemento successivo o precedente.
Conversione puntatori/interi
Convertendo un puntatore in un intero si rischia di perdere i 32 bit più significativi, poichè un puntatore usa 64 bit, mentre un intero 32.
Usare <stdint.h> per avere variabili di lunghezza indipendente dal tipo di architettura.
Come stampare puntatori
Utilizzare il formato `%p` insieme alla funzione printf().

Strutture e unioni

Strutture
Un tipo di dato composto che consente di raggruppare variabili di tipi diversi sotto un'unica entità.
`struct name { type1 v1; type2 v2; }`
Unioni
Una variabile come una struttura, ma sovrappone la memorizzazione dei campi. Può mantenere oggetti di diversa dimensione e tipo (in momenti diversi). L'occupazione di una union è pari all'occupazione del campo di dimensione massima.
Operatore . (punto)
Per accedere ad un campo di una struttura.
`struct_name.field`
Puntatori a strutture, operatore ->
Un puntatore può contenere l'indirizzo in memoria di una struttura. L'operatore `->` permette di accedere ai campi di una struttura puntata.
`(*pp).x` equivale a `pp->x`
Campi di bit nelle strutture
Permettono di scegliere il numero di bit che una variabile all'interno di una struct deve occupare.
Utilizza l'operatore `:`.
`struct p { int x : 2; }`
Assegnamento di strutture/unioni
Quando si assegna una struttura/unione, essa viene copiata interamente.
Costruttori di strutture
Sono funzioni speciali utilizzate per creare ed inizializzare variabili di tipo struttura con valori specifici.
Campi senza nome in strutture/unioni.
Si può dichiarare una struttura/unione senza specificare il nome dei campi.
Tipi incompleti di struttura/unione
Si può dichiarare una struttura/unione senza specificare i tipi dei campi. Questo permette a chi ne fa uso di scegliere i campi necessari.
Operatore offsetof
Data una struttura, `offsetof` indica quanti byte dista ogni campo dall'inizio della struttura.

Array

Array, operatore []
Un array è una collezione di elementi dello stesso tipo, organizzati in memoria in modo contiguo e accessibili tramite un indice numerico. L'operatore `[]` serve per accedere agli elementi in base al loro indice.
Definizione di array
La definizione di un array specifica il tipo degli elementi e la dimensione.
`tipo nome[dimensione]`
Array e puntatori
Togliendo le parentesi quadre si ottiene un puntatore al primo elemento dell'array. Se v[i] è di tipo T, v è di tipo T*. Se p è un puntatore e i è un intero: `p[i]` è un'abbreviazione di `*(p+i)`.
Inizializzazione di array
Inizializzazione statica: specificare i valori che andranno nelle posizioni occupate al momento della dichiarazione. `int array[3] = {1, 2, 3}`
Inizializzazione dinamica: assegnare un valore ad una posizione dell'array durante il programma.
`array[1] = 5`
Stringhe per inizializzare array
Un array di char può essere inizializzato al momento della dichiarazione usando una stringa racchiusa in doppi apici. In questo caso bisogna considerare uno spazio aggiuntivo per il carattere di terminazione stringa `'\0'`.
`char str[6] = "Hello"` -> `[H, e, l, l, o, \0]`
Array multidimensionali
Sono array in cui un elemento è a sua volta un array. L'accesso agli elementi di un array multidimensionale avviene specificando gli indici per ogni dimensione. `int matrice[3][4]`
Array di puntatori
Il tipo di un array può essere un puntatore. `int *array[3];`. Per accedere alla locazione puntata si dereferenzia la cella contenente il puntatore. `... *array[i] ...`

Funzioni

Funzioni: dichiarazione e definizione
La dichiarazione è composta dalla sola testata con tipo di ritorno, nome e parametri. La definizione è la vera e propria implementazione della funzione.
`tipo_ritorno nome_funzione(tipo_a arg_a)`
Passaggio per valore
Modalità di passaggio dei parametri che verranno modificati solo all'interno del corpo della funzione (variabili locali). Gli argomenti delle funzioni sono sempre passati per valore.
return
Istruzione per uscire dalla funzione. Se la funzione è di tipo void non si mette nessun parametro. Il compilatore deve conoscere il tipo del valore di ritorno delle funzioni.
Funzioni static
Una funzione visibile solo nel file sorgente in cui viene dichiarata. Si usa per fare incapsulamento.
Puntatori a funzione
Puntatori che contengono l'indirizzo di una funzione. Possono essere usati per chiamare la funzione puntata. Il puntatore è dato dal nome della funzione senza le parentesi tonde a seguirlo.
Array di dimensione variabile: come parametro, come variabili locali
Come parametro è solo rappresentativo scrivere `type v[]`, è obbligatoria la dimensione solo in

caso di array multidimensionali (la prima dimensione è tralasciabile).
Come variabili locali possono essere definiti usando una variabile per la dimensione, che verrè decretata a runtime.

Funzioni a numero variabile di parametri (variadiche)

Ci deve essere almeno un parametro definito e poi i tre puntini `...`.
Per definirle si usa l'header `stdarg.h` e 4 macro:
- va_list: dichiara una lista di parametri variabili.
- va_start: inizializza la scansione dei parametri di numero variabile.
- va_arg: prende un parametro alla volta dalla lista specificando il tipo richiesto.
- va_end: pulisce i parametri della lista.

Variabili

Variabili locali e globali

Tutto ciò che viene dichiarato al di fuori delle funzioni è globale, ossia visibile a tutti e permanente. Il resto è locale, ovvero visibile solo nel blocco in cui è dichiarato ed al termine dell'esecuzione del blocco viene rimosso.

Variabile static

Una variabile static locale mantiene il suo valore durante chiamate successive della funzione.
Una variabile static globale è visibile solo all'interno del file in cui è dichiarata e mantiene il suo valore per tutta l'esecuzione del programma.

Variabile extern

Variabile definita in un altro file sorgente (dove non deve essere static). Il linker si occuperà di risolvere la sua definizione in fase di collegamento. Permette di evitare errori di definizione multipla.

Variabile volatile

Variabile che può essere modificata da agenti esterni e quindi non viene ottimizzata dal compilatore.

Variabile register

Variabile che viene messa nei registri di CPU. Serviva quando una variabile era utilizzata spesso, con i compilatori moderni non serve a nulla.

const

Permette di definire costanti che non possono essere modificate dopo la loro inizializzazione.

Operatore cast (conversione esplicita di tipo)

Consente di modificare esplicitamente il tipo di una variabile.
`(nuovo_tipo) var`

Conversione automatica di tipo nelle espressioni

All'interno di un'espressione gli operandi possono essere convertiti nel tipo più "importante". La scala di importanza dei tipi (dal meno al più) è: char, short, unsigned short, int, unsigned int, long, unsigned long, float, double, long double.

Conversione automatica di tipo negli assegnamenti

Il tipo del dato da assegnare può essere modificato in automatico per essere coerente con il tipo della variabile. Stessa scala di importanza delle espressioni.

Conversione automatica di tipo nei parametri delle funzioni

Come per l'assegnamento, il tipo del parametro può essere modificato per matchare la dichiarazione della funzione. Stessa scala di importanza delle espressioni.

Avanzate

L'operatore sizeof
Restituisce la dimensione in byte di un tipo di dato o di una variabile.
enum
Definizione per dare delle regole a delle costanti. `enum` è un int, è solo uno strumento sintattico.
typedef
Permette di definire il nome di un tipo di dato.
`typedef type new_type`
goto e label
`goto` è un'istruzione per fare un salto non condizionato all'interno del codice ad una `label`, che è un identificatore univoco associato ad una posizione specifica nel codice.
Può essere utile a volte per non incasinare il codice.
Uso di !!
Usando `!!` si nega due volte un valore, ottenendo un booleano che rappresenta il valore di partenza.

Preprocessore

Preprocessore: costanti, macro, compilazione condizionale
Editor di testo che prepara il codice alla compilazione. Al suo interno è possibile definire:
- costanti: simboli definiti prima della fase di compilazione che verranno sostituiti da valori specifici nel codice sorgente `#define nome valore`
- macro: direttive che consentono di definire blocchi di codice e sequenze di istruzioni che verranno sostituite da valori specifici. Sono utili per creare codice flessibile e automatizzare compiti ripetitivi.
- compilazione condizionale: permette di includere o escludere porzioni di codice dalla compilazione basandosi su condizioni definite.
Macro di più linee
Per scrivere macro di più linee si usa `\` al termine della riga.
do while(0) - (statement expression/estensione gcc)
Un idioma utilizzato nel preprocessore per creare blocchi di codice che sembrano un'unica istruzione condizionale.
#if 0
Utilizzato per ignorare completamente una parte di codice durante la compilazione (come un commento).
Macro a numero variabile di parametri
Si utilizza l'header `stdarg.h` e si usa __VA_ARGS__ per catturare i parametri variabili.
#undef
Direttiva al preprocessore utilizzata per rimuovere una macro precedentemente definita.
#error e #warning
- `#error` : genera un errore di compilazione intenzionale che interrompe la compilazione e visualizza il messaggio di errore specificato.
- `#warning` : genera un avvertimento durante la compilazione, senza però interromperla.