Appunti informatica

Appunti informatica

Concetti fondamentali

Definiamo come **complessità computazionale** di un problema la complessità dell'algoritmo più efficiente in grado di risolvere tale problema; non è necessario esibire un simile algoritmo, è sufficiente dimostrare che un tale algoritmo esiste e che tale è la sua omplessità computazionale.

Informazioni generali sul C

La versione attuale di C è quella chiamata **C17** pubblicato nel 2018. In generale la storia del C parte negli anni 70 con lo sviluppo di *UNIX* con il piccolo linguaggio chiamato **B**. Nel 1973 venne chiamato C e poi 1989 divenne **C89** e poi con tutti i suoi successori chiamati **C90, C95, C99, C11 e C** ma in realtà queste informazioni sono superflue. Naturalmente il C è un linguaggio a basso livello perchè è molto vicino al linguaggio macchina. Inoltre il C è un linguaggio compilato che, a differenza di un linguaggio interpretato ha la necessità di avere un compilatore che trasforma il codice sorgente in azioni eseguibili per la macchina.
La particolarità del linguaggio C (ma anche di altri), è che sono **compilati**, ovvero eseguiti da un compilatore che traduce il codice sorgente in linguaggio macchina per ottenere un risultato. Inoltre, è un linguaggio **imperativo** (sequenziale), dove il concetto di oggetto ed istanza non esistono e dove tutti i problemi possono essere risolti con funzione **do, for if** ecc. Caratteristica importante del linguaggio sono i **puntatori**, ovvero variabili che contengono l'indirizzo di memoria di un'altra variabile permettendo di accedere direttamente a una licazione di memoria.
Il linguaggio presenta diversi pregi come:

- **Efficienza** data dai puntatori e dalla gestione della memoria;
- **Tipizzazione** dove ogni variabile ha il suo tipo ;
- **Basso livello** permette di produrre un codice finale più compatto ed efficiente;
- **Portabilità** permette lo sviluppo e il successivo passaggio da una macchina ll'altra,
  Le debolezze invece sono:
- Alta probabilitàdi errore nei programmi;
- Programmi difficili da capire a causa della flessibilità del linguaggio
- Difficoltà nella modifica perchè non è sempre considerata la manutenzione
  Per lo sviluppo ci si basa su un compilatore, delle **librerie run-time** e almeno un file sorgente. Ogni file sorgente viene compilato e reso file oggetto che, grazie a dei set di librerie eseguiti in contemporanea, servono ad avere un unico file eseguibile. Il file eseguibile è dato dal **Linker**,

ovvero un processo che unisce tutti i file oggetto (otetnuto dalla compilazione dei files), codice delle librerie esterne e **start-up code** (codice necessario al sistema operativo).

C sintassi

Nella header o, le prime righe vengono introdotte le librerie che si andranno ad usare durante il nostro programma. Per libreria definiamo un insieme di funzioni già stabilite che possono essere utilizzate semplicemnte richiamandole.
Sotto la dichiarazione delle variabili possiamo anche usare il comando **#define** che permette di definire sia delle costanti con ***DEFINE CONST 5*** oppure puoi dichiarare delle pseudofunzioni con il define. Queste funzioni si chiamano **Espressioni condizionali**, un esempio è: **DEFINE Max(a,b)(a > b ? a : b)**. Questa fuznione permette trovare il massimo valore tra due numeri casuali. Nella header di un codice sorgente trovaimo tutte istruzioni con carattere iniziale **#**, questo eprchè sono informazioni sviluppate dal **pre-processore**. Questo pre-processore controlla le prime righe di codice e, se trova una libreria, sostituisce la riga `#include <stdio.h>` con tutto il codice della libreria presente nelle directory. Se invece andassimo a mettere sotto le librerie questa definizione `#define Cifre 9` assegneremo alla stringa *Cifre* il valore 9. Un altro modo di dichiarare costanti è tramite il tipo **const** che, se usato nel main o anche come variabile globale, crea una variabile **read only** che non può essere modificata dall'utente finale.

Nella scrittura in C per la letterura da tastiera dobbiamo utilizzare la scnaf per prendere dati in input e, nel caso di caratteri si usa il getchar per il singolo carattere. Ecco alcune istruzioni base:

- **scanf**("%tipo", &nome_variabile).
- per la **printf** invece è: printf("testo %tipo", nome_variabile)
- simile alla printf abbiamo il **puts** che ha la stessa sintassi della printf ma manda a capo la stringa successiva.

Le operazioni di input, output ed errore sono gestite dai **canali standard (standard streams)**, ovvero tutti quei collegamenti tra il programma e l'ambiente operativo in cui viene eseguito. I cannali sono 3, ovvero

- **standard input stdin**
- **standard output stdout**
- **standard error stderr**

Un po' di comandi

- Il comando `system("PAUSE")` permette di avere l'output con **cmd** e non tramite il terminale dell'ide che si sta usando.
- `goto etichetta` permette di saltare pezzi di codice e di andare a una specifica etichetta (label) senza che vi sia una condizione effettiva.

- `return valore` permette di dare ocme ritorno un valore in base alla dichiarazione del tipo della funzione, queto perchè una funzione senza un return non può avere fine. Per esempio il main ha sempre bisogno di un return 0 perchè nella dichiarazione è di tipo int, quindi deve restituire un intero alla fine. Altre funzione, come quelle di tipo void, non hanno bisongo di un return.

Printf

Nella **printf** possiamo usare delle costanti aggiuntive per dare un formato alla stringa finale come ad esempio:
`printf("valore di n = %4d", n)` darà come risultato un valore allineato a destra di almeno 4 cifre. La stessa cosa ma con un - ci darà l'allineamento a sisnistra `printf("valore di n = %-4d", n)`.
Se invece andassimo a mettere questa notazione `printf("valore di n = %4.2d", n)` otterremo un output di almeno 4 caratteri e, nel caso di un intero, tanti zeri quanti ne servono per arrivare a 4 cifre. Con un float invece il valore dopo il punto indica il numero di cifre dopo la virgola.

Tipologie di dati e variabili

Ogni tipo può avere dei vincoli aggiuntivi per la rappresentazione del dato. Questi sono: **Long, short, signed e unsigned**.
Tutti i tipi di dati sono:

Tipo	Descrizione	Grandezza
%d	intero	2 Byte
%i	intero	2 Byte
short int %i	intero corto	2 Byte
long int %i	intero lungo	4 Byte
unsigned int %i	intero senza segno	2 Byte
unsigned long int %i	intero lungo senza segno	4 Byte
%c	char	1 Byte per carattere
unsigned char %c	carattere senza segno	1 Byte
%s	stringa	Dipende dalla grandezza
%f	float	4 Byte
%f	double	8 Byte
%lf	long double	10 Byte
%g	float senza 0 alla fine e in notazione scientifica	2 Byte
%o	intero ottale	2 Byte
%x	intero esadecimale	2 Byte

Nota bene che in C ogni float viene gestito come un double, per questo l'istruzione `float n = 4.5f` è un rafforzativo per la variabile stessa.

Una funzione utile per vedere quanti bit occorrono per i tipi oppure per una specifica variabile è **sizeof()**, che restituisce la grandezza in bit.

Abbiamo una divisione dei tipi di dato utilizzabili in C. I più importatnti sono quelli **scalari**, perchè i valori che li compongono sono distributi su una scala lineare, su cui si può stabilire una relazione di ordine totale. Questi tipo sono i puntatori, i tipi enumerativi e quelli aritmetici. Per definizione la **variabile** è l'individuazione di una certa area di memoria mediante un nome simbolico. Ogni variabile viene inserita nella RAM. Ogni variabile porta con se 2 valori, L-value e l'R-value. Il primo permette di vedere il valore dell'indirizzo di memoria ed è immutaible. Il secondo è il valore assegnato alla variabile che può cambiare

In informatica, **word (parola)** è un termine che identifica la dimensione nativa dei dati usati da un computer. Una word è un gruppo di bit di una determinata dimensione che sono gestiti come unità da un
microprocessore. La dimensione (o lunghezza) della word è un'importante caratteristica dell'architettura di un computer. Le architetture dei PC sono delle word, e possono essere a 32 o 64 bit. Queste avranno gli stessi registri ma con grandezze diverse, così come la grandezza delel variabili sarà diversa.

Per informazioni sull'ASCII visita le dispense di architetture degli elaboratori

In C non esiste il tipo boleano (boolean), per questo un semplice controllo su un flag va fatto con if(x) dove si controlla il valore di x e, in caso ci fosse un numero diverso da 0 la condizione si verifaca, altrimenti restituisce un valore falso.

Quando parliamo di **variabili** dobbiamo tenere presente la zona di codice dove queste vengono inizializzate e usate. Queste variabili lavorano tramite delle regole che permettono a ogni variabile di lavorare su una determinata porzione di codice e si chiamano **scope**. Lo scope permette di poter inizializzare più variabili con lo stesso nome in parti di codice diverso. Nel caso di variabili locali possiamo creare più variaibili con stesso nome in funzioni diverse. Attenzione che non è possibile fare la stessa cosa con variaibili globali. Ogni variabile viene memorizzata sullo stack di memoria che permette alla CPU di lavorare in fase di running. Questo avviene solo con variabili editabili, ma se usassimo una varibile di tipo static, ovvero che non può cambiae di valore, questa verrà inserita in una *locazione di memoria permanente* per tutta al durata del programma.
Una variaibile globale può essere utilizzata anche su altri file se si aggiunge la parola **extern** `extern int Valore`.

Operatori

Molti sono gli operatori in C che possono migiliorare la lettura e scrittura del ocdice. I più comuni sono:

Tipo	Descrizione
+=	addizione del valore a sinistra con quello a destra
-=	sottrazione del valore a sinistra con quello a destra
*=	moltiplicazione del valore a sinistra con quello a destra
/=	divisione del valore a sinistra con quello a destra
++a	incremento di 1 prefisso e restituisce il valore
a++	restituisce e incrementa di 1 postfisso
--a	decremento di 1 prefisso e restituisce il valore
a--	restituiesce e decrementa di 1 postfisso
^	Xor logico
~	complemento a 1
>>	shift a destra del primo operando per un numero di bit pari al valore del secondo `x >> 3`
<<	shift a sinistra del primo operando per un numero di but paru al valore del secondo `x << 4`

Se gli incrementi o decrementi sono l''unica operazione di una riga di comando lavorano allo stesso modo, ovvero che il valore sarà sempre incrementato o decrementato.

Casting

Il **casting** permette di passare una variabile di un tipo a un altro, ma questo non vuol dire che avrà lo stesso valore per l'utente finale. La funzione di casting inoltre permette di svolgere operazioni in modo più flessibile anche se non si hanno certi tipi di dato. Inoltre, il cacsting segue una gerarchia, ovvero che se in un'operazione andiamo a castare degli interi in float, naturalmente tutti gli interi diventeranno float. La gerarchia è ***int < float < double < long double***. `float avg = nHrs/(float)nDays;` In questo caso tutto diventa float.

Sistemi di calcolo

La macchina di Turing universale (un modello) definisce l'insieme dei problemi calcolabili. La **Turing equivalneza** è la proprietà dei modelli di calcolo che hanno lo stesso potere computazionale di una macchina di Turing.

La programmazione si basa su strutture di controllo, le 3 fondamentali sono la *concatenazione, la selezione o condizione e l'iterazione*. Queste strutture permettono la scrittura facilitata del codice, ma non cambia la potenza della macchina di Turing.

I cicli

Abbiamo diversi modi di fare dei cicli iterativi in C, ma in generale nei linguaggi di programmazione. Il primo è il costrutto `while(espressione)` che controlla che l'espressione al suo interno sia vera o

falsa. In caso fosse vera il while permette di far eseguire il suo blocco di codice sottostante, altrimenti in caso fosse falsa farebbe saltare quel blocco di codice.
Il `Do{codice}while(espressione)` a differenza del while permette l'esecuzione del codice almeno una volta e, avendo il while alla fine del blocco di codice, si controllerà che l'espressione sia vera o falsa. Come prima in caso fosse vera torna a eseguire tutto il blocco di codice sovrastante e controlla nuovamente nel while, altrimenti esce dal ciclo e continua fino a terminare.
Il ciclo for è invece una rappresentazione di un do-while compatta, dove sono presenti la dichiarazione di un contatore, l'espressione di controllo e un operazione sulla sul contatore. Questo permette al for di essere un ciclo *finito*, ovvero dove si saprà da subito il numero di iterazioni massime, a differenza di un while che può avere infinite iterazioni. La sintassi del for è `(contatore, condizione, incremento)` `quindi (for int i = 0; i < n; i++)` dove la dichiarazione interna di i non è necessaria se già dichiarata fuori dal for.
Il ciclo for viene eseguito in un modo specifico, ovvero per prima si legge l'espresisone del for, poi il blocco di codice all'interno del for e poi si incrementa controlla che sia ancora vera la condizione.
Esistono anche for che lavorano con due variabili contemporaneamente, ad esempio `for(int i = 0, j = 0; i + j < 100; i++, j++)`.

Nei cicli possiamo trovare 2 istruzioni facoltative, ovvero la **break e la continue**. La prima se messa in un qualsiasi ciclo fermerà il ciclo stesso, saltando tutte le istruzioni successive. Con il continue invece torneremo all'inizio del ciclo, saltando sempre tutte le informazioni sottostanti al continue.

Vettori

Un vettore è una variabile aggregata in grado di contenere un certo numero di dati **tutti dello stesso tipo**. La dichiarazione di un vettore di interi è `int vettore[100]` dove viene dichiarata assieme al nome simbolico anche la grandezza massima di *celle* del vettore. Un array può anche essere dichiarato così : `int br[5] = {1,2,3,4,5}`Questo però vuol dire che quando andremo a lavroare con il nostro vettore partiremo dalla cella 0 e arriveremo fino all cella N - 1 perchè lo 0 è compreso. Questi sono vettori di tipo monodimensionale, a differenza dei multidimensionali che sono delle matrici come ad esempio `int M[10][5]` dove avremo una matrice di 10 righe e 5 colonne. **MATRICI DA SPIEGARE PAGINA 33**

DA FARE LUCUDI 8