Elementi di Base del Linguaggio C

Struttura Generale di un programma (parte dichiarativa e esecutiva con terminazione istruzioni)

Un programma e' formato da una **parte dichiarativa** e una **parte esecutiva**.
La parte dichiarativa e' quella in cui vengono riservate (**dichiarate**) le variabili, quella esecutiva contiene le istruzioni che l'esecutore deve svolgere. Per ora considereremo che sia la parte dichiarativa sia quella esecutiva sono contenute in una funzione chiamata `main` che viene eseguita automaticamente quando viene lanciato il programma. Vedremo in seguito cosa si intende per funzione, per ora ci basta considerare la funzione `main` che in pratica racchiude tutto il nostro programma.
Quindi per ora i nostri programmi avranno la forma seguente:

```
int main()
{
     /* blocco dichiarativo */

     /* inizio blocco esecutivo */
     ...
     return 0;
     /* fine blocco esecutivo */
}
```

Capiremo in seguito il significato della parola `int`. Invece, e' importante sapere che ogni funzione deve sempre avere le parentesi tonde dopo il suo nome, e tutto il suo contenuto deve essere racchiuso tra parentesi graffe.
Racchiuderemo i commenti, cioe' le frasi che non fanno parte del linguaggio di programmazione ma servono per rendere piu' leggibile il codice sorgente, tra i caratteri `/*` e `*/`. Un commento puo' anche occupare piu' di una riga.
In C il blocco dichiarativo precede il blocco esecutivo, cioe' in generale non e' possibile dichiarare una variabile dopo una istruzione. Il blocco esecutivo del `main` finisce sempre con l'istruzione `return 0;` che comprenderemo piu' avanti. Ogni istruzione termina con un punto e virgola.
Si chiama **blocco** un insieme di istruzioni racchiuso tra parentesi graffe.
Attenzione: il C e' un **linguaggio case-sensitive**, cioe' distingue le lettere maiuscole e minuscole. Scrivere `Main` invece che `main` da' luogo ad un errore di compilazione.

Tipi di Dato Elementari Predefiniti

I tipi di dato elementari predefiniti sono i seguenti:

Nome del Tipo	***Descrizione***	***Range***
`char`	carattere	
`int`	numero intero	
`float`	numero decimale (8 cifre decimali)	
`double`	numero decimale con precisione doppia (16 cifre decimali)	

E' possibile premettere al nome del tipo un *modificatore* (per esempio, `unsigned`, `short`, `long`, ecc), per ottenere altri tipi.

La dimensione in memoria di una variabile di un certo tipo dipende dalla piattaforma (hardware+sistema operativo), con questi vincoli:

- un `char` occupa meno spazio di un `int`
- un `int` occupa meno spazio di un `float`
- un `float` occupa meno spazio di un `double`

Tutte le variabili dello stesso tipo occupano lo stesso spazio in memoria nella stessa piattaforma. L'intervallo (**range**) di valori che una variabile di un certo tipo puo' assumere dipende dallo spazio che essa occupa in memoria. Per esempio, se uso 16 bit per rappresentare un intero, e ogni bit puo' avere valore 0 oppure 1, posso ottenere 2^{16}=65... valori. In genere questi valori sono centrati nel valore 0, e quindi il range degli interi e' compreso tra -.... e .... Se invece uso il modificatore unsigned il range sara' compreso tra 0 e ....
In generale, una variabile che occupa N bit avra' valori compresi tra -2^{N-1} e $2^{N-1}-1$.
Esiste un operatore, `sizeof`, che permette di conoscere lo spazio occupato da un tipo o da una particolare variabile. Per esempio, per conoscere lo spazio occupato da una variabile intera, potremo valutare l'espressione `sizeof(int)` oppure `sizeof(a)` se `a` e' una variabile ti tipo intero.
Non esiste il tipo booleano, e quindi al suo posto si usera' un intero con valore 1 per indicare VERO e valore 0 per indicare FALSO.

Dichiarazione delle Variabili

La dichiarazione delle variabili deve avvenire all'inizio di un blocco, e ha la seguente sintassi:

```
<nomeTipo> <nomeVariabile>;
```

Per esempio:

```
int a;
char x;
int a,b,c;
```

Piu' variabili dello stesso tipo possono essere dichiarate nella stessa riga, e in questo caso i loro nomi devono essere separati da virgole. In genere i nomi delle variabili vengono scritti tutti minuscoli. Poiche' gli identificatori delle variabili non possono contenere spazi bianchi, per indicare nomi composti, e' possibile seguire due convenzioni:

- separare una parola dall'altra usando un carattere _ (esempio: `anno_di_nascita`)
- iniziare ogni parola che segue la prima con lettera maiuscola (esempio: `annoDiNascita`)

Assegnamento

L'operatore di assegnamento e' dato da un segno di uguale:

Linguaggio di progetto	***Linguaggio C***
<--	=

Esempi di assegnamenti sono i seguenti:

```
a = 10;
b = a;
c = b = a;
```

La catena di assegnamenti dell'ultima riga ha senso perche' in C l'assegnamento e' una espressione che ha un suo valore, e cioe' il valore assegnato. Quindi l'espressione:

```
b = a;
```

viene ha un suo valore, dato da `a`.
In una catena, gli assegnamenti vengono eseguiti e quindi valutati da sinistra verso destra.
Quindi quando trova l'istruzione:

```
c = b = a;
```

l'esecutore opera in questo modo:
- esegue l'assegnamento `b = a`
- lo valuta e trova il valore `a`
- assegna questo valore a `c`

Inizializzare una variabile significa assegnarle un valore per la prima volta.
L'inizializzazione puo' anche avvenire nella stessa riga in cui la variabile viene dichiarata:

```
int a = 10;
int b = a, c = 4, d = -20;
```

In caso di piu' dichiarazioni nella stessa riga, ognuna puo' essere inizializzata usando sempre la virgola come carattere di separazione.

Casting

Qualche volta puo' essere utile considerare una variabile o un valore di un tipo come appartenente ad un altro tipo. In questo caso si deve indicare prima di esso, racchiuso tra parentesi tonde, il tipo che si vuole considerare, secondo la sintassi:

```
(<tipo>) <valore | variabile>
```

Per esempio:

```
(float)10
(int)x
```

Operatori (aritmetici, relazionali, logici)

Gli **operatori aritmetici** sono i seguenti:

Operatori Aritmetici	
`+`	addizione
`-`	sottrazione
`*`	moltiplicazione
`/`	divisione
`%`	resto della divisione

E le loro reciproche precedenze sono le stesse che avete studiato in Matematica: moltiplicazioni e divisioni nell'ordine in cui si incontrano, addizioni e sottrazioni nell'ordine in cui si incontrano.

Non esiste un operatore specifico per l'operazione di potenza, ma esiste una funzione che vedremo in seguito.
C'e' da specificare qualcosa che riguarda la divisione: in C esiste un unico operatore per la divisione tra interi (con eventuale calcolo del resto) e la divisione tra numeri decimali. L'esecutore capisce quale deve eseguire sulla base del tipo dei suoi operatori. Per esempio:

```
float x, y, ris;
x = 15;
y = 10;
ris = x/y;
```

In questo caso `ris` avra' valore 1.5 perche' gli operatori della divisione, pur avendo valori interi, sono di tipo `float`.
Invece nel caso:

```
int x, y, ris;
x = 15;
y = 10;
ris = x/y;
```

`ris` avra' valore 1 e se si calcola anche il resto attraverso l'operazione `x%y` si otterra' il valore 5.
Oppure si puo' operare attraverso il cast:

- `5/10` ha valore 0
- `(float)5/(float)10` ha valore 0.5

Gli **operatori relazionali** sono quelli che servono per confrontare due valori numerici e sono i seguenti:

Operatori Relazionali	
<	minore
<=	minore o uguale
>	maggiore
>=	maggiore o uguale
==	uguale
!=	diverso

Bisogna stare molto attenti a non confondere l'operatore di confronto `==` con l'operatore di assegnamento `=`

Gli **operatori logici** sono i seguenti:

Operatori Logici	
!	NOT
&&	AND
\|\| | OR |

Operatore di indirizzamento &

Vedremo tra poco che talvolta e' necessario potere ottenere l'indirizzo di una particolare variabile. Questo e' possibile applicando alla variabile l'operatore di indirizzamento `&` che ha la seguente sintassi:

```
&<nomeVariabile>
```

per esempio:

```
&x
```

rappresenta l'indirizzo della variabile `x`.

Operazioni di Input / Output

Come gia' abbiamo detto per il `main`, anche le operazioni di input e di output in C sono delle funzioni. Capiremo meglio cosa vuol dire in seguito, per ora limitiamoci a considerarle delle 'scatole nere' che in qualche modo che a noi non interessa svolgono il compito che ci serve.
L'operazione di output e' svolta dalla funzione `printf`, che ha la seguente sintassi:

```
printf(<stringa di formattazione>, [<lista dei valori da stampare in output>]);
```

La **stringa di formattazione** e' una stringa che contiene i valori costanti che devono essere stampati in output e alcune sequenze di caratteri speciali che servono a comunicare all'esecutore che deve stampare in output i valori di una o piu' espressioni oppure che deve compiere particolari azioni (andare a capo, lasciare uno spazio di tabulazione, ecc). La stringa di formattazione, come tutte le stringhe in C, e' delimitata da doppi apici.
Gli elementi racchiusi tra parentesi tonde nella sintassi della funzione sono i suoi **parametri** o **argomenti**, e quelli racchiusi tra parentesi quadre sono opzionali, cioe' possono non essere presenti. Infatti, se la stringa di formattazione non contiene caratteri speciali e si devono stampare solo valori costanti, non si deve indicare nessun ulteriore parametro.
Torniamo alla stringa di formattazione: i caratteri speciali che puo' contenere sono gli specificatori di formato oppure le sequenze di escape.
Gli **specificatori di formato** (**format specifiers**) sono costituiti da un carattere `%` seguito da particolari caratteri, tra cui i principali sono: `d`, `c`, `f`,`%`, il cui significato e' il seguente:

Specificatore di Formato	***Significato***
`%d`	un numero intero
`%c`	un carattere
`%f`	un numero decimale
`%%`	il carattere %

Notate l'ultimo simbolo: dato che il carattere % e' un carattere speciale, come si fa a dare in output proprio il carattere %, senza rischiare che venga confuso con l'inizio di uno specificatore di formato? Non c'e altro modo se non quello di utilizzare un particolare specificatore di formato e cioe' %%.
Le **sequenza di escape** (**escape sequences**) principali sono:

Sequenza di Escape	***Significato***
`\n`	andata a capo (*newline*)

Sequenza di Escape	***Significato***
`\t`	carattere di tabulazione (*tab*)
`\"`	doppi apici
`\\`	il carattere \

Le ultime due sequenze di escape hanno un significato analogo allo specificatore di formato %% che abbiamo gia' incontrato: servono per stampare in output i caratteri " e \ senza che vengano confusi rispettivamente con la terminazione della stringa di formattazione o con l'inizio di una sequenza di escape.
Tutti gli altri caratteri che compaiono nella stringa di formattazione e che non fanno parte dei caratteri speciali vengono stampati in output cosi' come sono.
Esaminiamo alcune semplici stringhe di formattazione:

"ciao"
"ciao\n"
"Piero e' nato nell'anno %d e quindi ora ha %d anni"
"%f + %f = %f"

Le prime due fanno stampare in output la parola *ciao*, e la seconda va a capo dopo averla stampata. Non richiedono ulteriori parametri.

La terza serve per comunicare all'esecutore i seguenti ordini:
- stampa in output la frase *Piero e' nato nell'anno*
- stampa il valore intero che troverai nella variabile indicata come argomento successivo
- stampa in output la frase *e quindi ora ha*
- stampa il valore intero che troverai nella variabile indicata come argomento successivo
- stampa in output la parola *anni*

Richiede altri due parametri, entrambi di tipo intero.

La quarta stringa di formattazione serve per stampare in output i valori di tre variabili `float` rispettivamente come operandi e risultato di una addizione. Richiede altri tre parametri, tutti di tipo intero.

Esaminiamo quindi le istruzioni di output che possono corrsipondere alle precedenti stringhe di formattazione:

```
printf("ciao");
printf("ciao\n");
printf("Piero e' nato nell'anno %d e quindi ora ha %d anni",
       annoNascita, eta);
printf("%f + %f = %f", x, y, z);
```

Notate che i diversi parametri della funzione `printf`, come quelli di tutte le funzioni, sono separati da virgole.
Come abbiamo gia' detto, i parametri successivi alla stringa di formattazione possono essere delle espressioni generiche. Per esempio, l'istruzione:

```
printf("Il doppio di %d e' %d", a, 2*a);
```

stampa il valore della variabile `a` e del suo doppio.

L'operazione di input e' svolta dalla funzione `scanf`, che ha la seguente sintassi:

```
scanf(<stringa di formattazione>, [<lista delle variabili in cui memorizzare i valori presi in input>]);
```

Questa funzione si serve della stringa di formattazione per sapere quanti valori deve acquisire e il tipo di ciascun valore. Deve necessariamente avere almeno un altro parametro che indica la zona di memoria dove dovra' essere memorizzato il valore acquisito, per ogni valore da acquisire. Perche' la funzione possa effettivamente accedere a questa zona di memoria, deve conoscere il suo indirizzo, che si ottiene da una variabile attraverso l'operatore di indirizzamento che abbiamo gia' visto.
Un possibile esempio di utilizzo della funzione `scanf` e' il seguente:

```
int a;
float b;
char c;
scanf(“%d %f %c”, &a, &b, &c);
```

Vedremo tra poco che alcune variabili indicano gia' di per se' un indirizzo, e in quel caso non si deve usare l'operatore di indirizzamento.

Quando si acquisiscono dati dalla tastiera, bisogna tenere conto del carattere di andata a capo che si immette per terminare l'operazione di input. Quel carattere rimane nel canale di input dell'esecutore, in attesa di essere letto in seguito, e questo puo' generare errori.
Per esempio, se si fornisce in input il numero 10 e poi si preme il tasto invio, il numero 10 verra' memorizzato in una variabile opportuna e il carattere di andata a capo resta nel canale di input. Se poi l'esecutore esegue un'altra operazione di input nel quale e' richiesto di inserire un carattere, non aspettera' che venga inserito un altro carattere, ma leggera' l'andata a capo che e' rimasta in sospeso. Presumibilmente, questo non e' il comportamento desiderato dal progettista. Come si evita questo problema? Includendo i caratteri:

```
%*c
```

nella stringa di formattazione ogni volta che si presume venga premuto il tasto invio, per esempio alla fine.
I caratteri `%c` significano che l'esecutore legge dall'esterno un carattere; il carattere `*` indica all'esecutore che questo carattere non deve essere assegnato ad una variabile ma deve essere semplicemente scartato.
Per esempio, l'istruzione:

```
scanf(“%d%*c”, &x);
```

legge un numero intero e lo memorizza nella variabile `x`. Inoltre, elimina dal canale di input (talvolta chiamato **buffer**) il carattere di andata a capo.