CORSO DI PROGRAMMAZIONE
A.A. 2014-15

Dispensa 2

Laboratorio

Dott. Mirko Ravaioli
e-mail: mirko.ravaioli@unibo.it

2.1 La direttiva #include

La direttiva #include ordina al compilatore C di inserire il contenuto di un file di inclusione nel programma in fase di compilazione. Un *file di inclusione* è un file che contiene le informazioni richieste dal programma o dal compilatore. Sono molti i file di questo tipo (detti anche *file di intestazione*) supportati dal compilatore. Non è mai necessario modificare le informazioni contenute in questi file, ed è per questo che vengono mantenuti separati dagli altri file. I file di inclusione dovrebbero avere tutti l'estensione .H (ad esempio stdio.h).

La direttiva #include significa "aggiungi al programma il contenuto del file .h"

2.2 La funzione puts()

Per visualizzare i messaggi testuali è possibile utilizzare la funzione puts(), anche se quest'ultima non può stampare valori numerici delle variabili. La funzione puts() richiede come argomento una sola stringa che viene visualizzata con l'aggiunta automatica dei un "a capo".
Ad esempio l'istruzione:

```
puts("Ciao a tutti");
```

svolge lo stesso compito di

```
printf("Ciao a tutti\n");
```

Nella stringa passata alla funzione puts() è possibile specificare le sequenze di escare che avranno lo stesso effetto di quelle relative alla funzione printf(). La funzione è inclusa nella libreria stdio.h.

2.3 La funzione exit()

Questa funzione termina il programma e restituisce il controllo al sistema operativo; richiede un solo argomento di tipo intero che viene passato al sistema operativo per indicare il successo o il fallimento del programma. Sintassi:

```
exit(status);
```

Se *status* vale 0, indica che il programma è terminato normalmente, Il valore 1 indica che il programma è terminato con un errore. Il valore restituito è solitamente interrogato da sistema operativo. Per utilizzare exit() il programma deve includere il file stdlib.h che definisce anche due costanti simboliche da utilizzare come argomenti:

```
#define EXIT_SUCCESS   0
#define EXIT_FAILURE   1
```

2.4 La memoria del Computer

I computer utilizzano la memoria ad accesso veloce (RAM) per archiviare le operazioni su cui lavorano. La RAM (Random Access Memory) è composta da circuiti integrati che si trovano sulla scheda madre del computer e le informazioni in essa salvate vengono cancellate ogni volta che si spegne il computer. Da qui la definizione che la RAM è una memoria "volatile". E' possibile riscrivere le informazioni all'interno della RAM ogni volta che so vuole. Ogni computer ha un certo quantitativo di memoria installata, la dimensione si misura in kbyte (Kb). Il byte è l'unità di misura fondamentale delle memorie informatiche. Per farsi un idea del quantitativo di memoria necessario all'archiviazione di alcune tipologie di dati ecco alcuni esempi:

Dato	***Byte Richiesti***
La lettera "b"	1
Il numero 450	2
il numero 14.3124	4
La frase "il mondo non crea eroi"	22
Una pagina di testo	Circa 3000

La RAM di un computer è organizzata in modo sequenziale: quindi ogni byte è messo si seguito all'altro. Ogni byte ha un indirizzo unico attraverso cui viene identificato, che lo distingue da tutti gli altri byte della memoria. Gli indirizzi vengono assegnati dalle locazioni di memoria di ordine progressivo, cominciando dall'indirizzo 0.

La RAM ha diversi utilizzi in particolare per quanto ci riguarda essa permette di memorizzare i dati. I dati sono le informazioni su cui lavorano i programmi: tutte le informazioni che il programma deve eseguire vengono memorizzate all'interno della RAM durante l'esecuzione.

2.5 Le variabili

Una *variabile* è un nome simbolico che si assegna a una allocazione di memoria utilizzata per la memorizzazione dei dati. Le variabili devono essere definite prima di essere utilizzate; la definizione di una variabile indica al compilatore il nome della variabile e il tipo di dati che è destinata a contenere.

2.5.1 I nomi delle variabili

I nomi delle variabili in C devono seguire le seguenti regole:

- I nomi possono contenere lettere, numeri e il trattino di sottolineatura (_)
- I primo carattere di un nome DEVE essere una lettera. Anche il trattino di sottolineatura può essere utilizzato come primo carattere, ma dato che in questo caso la variabile viene trattata in modo particolare dal compilatore, per il momento è il caso di non utilizzarlo come carattere iniziale.
- Le lettere maiuscole e minuscole sono trattate come entità differenti. Quindi i nomi *pippo* e *Pippo* si riferiscono a due variabili differenti.
- Le *parole chiave* del C on possono essere utilizzate come nomi di variabili. Le parole chiave sono istruzioni del linguaggio C.

Ecco degli esempi corretti e non di nomi di variabili:

Nome variabile	
Numero	valido

stipendio_mensile	valido
g7t7_rfg7	valido
_1978_compleanno	valido ma sconsigliato
testo#progetto	Sbagliato: contiene il carattere #
double	Sbagliato: è una parola chiave del C
9vincitori	Sbagliato: il primo carattere è un numero

In C generalmente si utilizzano nomi di variabili scritte tutte in minuscolo (anche se non è richiesto dal linguaggio); normalmente i nomi scritti in maiuscolo sono riservati alle costanti.

Fate molta attenzione in quanto la stessa lettera in maiuscolo e minuscolo viene considerata diversa.

Per la maggior parte dei compilatori i nomi delle variabili non possono superare i 31 caratteri; anche se è possibile utilizzare nomi più lunghi il compilatore considera solo i primi 31. Il nome della variabile aiuta a rendere più chiaro il suo utilizzo all'interno del programma per il programmatore. Al compilatore non fa differenza avere una variabile con nome y o stipendio_mensile, nel primo caso però il significato della variabile non sarebbe stato così chiaro per chi avesse letto il codice sorgente come invece accade nel secondo caso.

2.5.2 Tipi di variabili

I valori che un programma può memorizzare possono essere molto diversi e quindi richiedono spazi in memoria diversi per poterli memorizzare.

Il C prevede un numero ristretto di tipi di dati fondamentali:

Tipo di dato	***Dimensione***	***Descrizione***
char	1 byte	contiene un carattere dell'ambiente in cui risiede il compilatore
int	2 byte*	un intero, solitamente pari alla dimensione naturale degli interi sulla macchina in cui risiede il compilatore, quindi non è detto che la dimensione sia pari a 2 byte
float	4 byte	numero con virgola mobile, con precisione singola
double	8 byte	numero con virgola mobile, con precisione doppia

Rappresentazione a virgola fissa: Dato che in un bit non è rappresentabile la virgola il metodo più semplice per rappresentare numeri frazionari è quello di scegliere arbitrariamente la posizione della virgola (ad es. se si sceglie di usare 4 bit per la parte intera e 4 per la parte frazionaria)

Rappresentazione in virgola mobile: Esistono innumerevoli modi per rappresentare numeri in virgola mobile ma il sistema più utilizzato è lo standard IEEE P754; questo metodo comporta l'utilizzo della notazione scientifica, in cui ogni numero è identificato dal segno, da una mantissa e dall'esponente.

In aggiunta è possibile qualificare questi tipi di dati in vari modi utilizzando i prefissi:

- short
- long
- unsigned

Gli attributi *short* e *long* si applicano solo agli interi. L'idea è che *short* e *long* denotino interi di diverse lunghezze. Dove è possibile *int* è invece di solito la grandezza naturale di riferimento per una macchina.

L'attributo *unsigned* (privo di segno) può essere applicato a tutte le tipologie, i numeri *unsigned* sono sempre positivi o al più nulli. il tipo *long double* denota numeri con virgola mobile a precisione multipla, come per gli interi al dimensione degli oggetti è definita dall'implementazione. Tutte le variabili per default sono di tipo *signed*.

Tipi di dati supportati:

Tipo Variabile	***Parola chiave***	***Byte***	***Intervallo***
Carattere	`char`	1	da -128 a 127
Intero*	`int`	2	da -32768 a 32767
Intero corto*	`short int` (o anche solo: `short`)	2	da -32768 a 32767
Intero lungo*	`long int` (o anche solo: `long`)	4	da -2147483648 a 2147483647
Carattere senza segno	`unsigned char`	1	da 0 a 255
Intero senza segno	`unsigned int`	2	da 0 a 65535
Intero lungo senza segno	`unsigned long int` (o anche: `unsigned long`)	4	da 0 a 4294967295
Variabile mobile con precisione singola	`float`	4	da 1.2E-38 a 3.4E308
Variabile mobile con precisione doppia	`double`	8	da 2.2E-38 a 1.8E308[2]

*dipende dalla macchina

La dimensione dei tipi di dato può variare a seconda della piattaforma utilizzata, lo standard ANSI ha definito cinque regole che ogni compilatore è tenuto a rispettare a prescindere dalla macchina su cui opera:

1. La dimensione di un *char* è sempre un byte
2. La dimensione di un *short* è minore o uguale a quella di un *int*
3. La dimensione di un *int* è minore o uguale ad un *long*
4. La dimensione di un *unsigned* è uguale a quella di un *int*
5. La dimensione di un *float* è minore o uguale a quella di un *double*

2.5.3 Dichiarazione di una variabile

Prima di poter utilizzare una variabile in un programma C è necessario dichiararla. La dichiarazione indica al compilatore il nome e il tipo di una variabile e opzionalmente il valore di inizializzazione da assegnare. Se il programma tenta di utilizzare una variabile non dichiarata il compilatore genera un errore.

La sintassi per dichiarare una variabile:

```
tipoDato nomeVariabile;
```

Dove con *tipoDato* si specifica il tipo di variabile e deve essere una delle parole chiave definite nel paragrafo precedente. Il *nomeVariabile* indica il nome della variabile e deve rispettare le regole descritte in precedenza.

Alcuni esempi di dichiarazione di variabili:

```
int annoNascita;
```

```
char iniziale;
int numeroScarpa, anni;
double percentuale;
float valore;
int conta, numero, inizio;
```

Alcuni esempi di dichiarazione con gli attributi:

```
unsigned int valoreNumerico;
unsigned char carattere;
unsigned double percentualePositiva;
long int valore;
unsigned long int valoreIntero;
```

2.5.4 Assegnare un valore ad una variabile

Quando si dichiara una variabile si chiede al compilatore di riservare dello spazio in memoria per la stessa. Il valore presente in questo spazio non è prevedibile, potrebbe essere qualsiasi cosa. Quindi è bene associare alla variabile un valore certo prima di utilizzarla. Questa operazione può essere fatta in diversi modi: dopo la dichiarazione o contemporaneamente alla dichiarazione.

```
int annoNascita;
annoNascita = 1978
```

L'operatore uguale (=) serve per associare un valore alla variabile. La variabile è sempre posta a sinistra dall'operatore. In C il significato di *annoNascita = 1978* significa: **assegnare il valore 1978 alla variabile di nome *annoNascita*** e NON "annoNascita è uguale a 1978".

Per dichiarare una variabile durante la dichiarazione:

```
int annoNascita = 1978;
double percentuale = 0.01;
double tasso = 22.4;
```

Bisogna fare attenzione a non inizializzare una variabile con un valore al di fuori dell'intervallo consentito dal tipo. Ecco per esempio due dichiarazioni non valide:

```
int peso = 1000000000000;
unsigned int valore = -2132;
```

Attenzione perchè il compilatore C non individua questo tipo di errori, il programma potrebbe essere compilato e linkato senza errori ma potrebbe dare risultati inaspettati in fase di esecuzione.

2.6 La funzione sizeof()

La funzione restituisce il numero di byte necessari per il tipo di dato passato come argomento alla funzione. La sintassi della funzione:

```
sizeof(tipoDato);
```

Un esempio di utilizzo:

```
#include <stdio.h>

int main()
{

    printf("\n numero intero (int) byte: %d", sizeof(int));
    printf("\n numero intero lungo (long int) byte: %d", sizeof(long int));
    printf("\n carattere (char) byte: %d", sizeof(char));
    printf("\n numero con virgola (float) byte: %d", sizeof(float));
    printf("\n numero con virgola (double) byte: %d", sizeof(double));

    return 0;
}
```

Il programma una volta compilato ed eseguito darà come risultato:

```
numero intero (int) byte: 4
numero intero lungo (long int) byte: 4
carattere (char) byte: 1
numero con virgola (float) byte: 4
numero con virgola (double) byte: 8
```