Il linguaggio C - Introduzione

- Il **C** è un linguaggio **imperativo** legato a Unix, adatto all'implementazione di compilatori e sistemi operativi.
- È stato progettato da D. Ritchie per il PDP-11 (all'inizio degli anni '70). Nel 1983 l'**ANSI** ne ha definito una versione standard **portabile** (ANSI C).
- A differenza dei linguaggi da cui ha tratto le idee fondamentali, ovvero, BCPL (M. Richards) e B (K. Thompson), è un linguaggio **tipato**.
- Il C è **compilato**; la compilazione è preceduta da una fase di *preprocessing* (sostituzione di macro, inclusione di file sorgenti ausiliari e compilazione condizionale).
- Il C è considerato un linguaggio ad *alto livello, ma non "troppo"* in quanto fornisce le primitive per manipolare numeri, caratteri ed indirizzi, ma non oggetti composti come liste, stringhe, vettori ecc.
- Il C è un linguaggio "piccolo": non fornisce direttamente nemmeno delle primitive di input/output. Per effettuare queste operazioni si deve ricorrere alla **Libreria Standard**. Si può pensare al C come al nucleo imperativo di Java più i **puntatori** e la gestione a **basso livello** di numeri, caratteri e indirizzi.

Struttura di un programma C

Consideriamo il programma C che stampa a video la stringa `ciao, mondo!` seguita da un avanzamento del cursore all'inizio della linea successiva:

```
#include <stdio.h>

main()
{
    printf("ciao, mondo!\n");
}
```

- Ogni programma C è composto da variabili e funzioni (contenenti comandi); fra queste ne esite una particolare, chiamata `main`, da cui **inizia l'esecuzione** e che quindi deve essere presente in ogni programma. Le due parantesi `()` vuote dopo il `main` significano che quest'ultimo non prende alcun parametro in input.

- La prima riga è una **direttiva al preprocessore** che dice di includere le funzioni per l'input/output della libreria standard prima di compilare il programma.

- La funzione `printf` della libreria standard stampa a video (standard output) la stringa fornita come parametro. All'interno di quest'ultima la **sequenza di escape** `\n` specifica il carattere speciale di "avanzamento all'inizio della linea successiva" o newline.

Compilazione di programmi C

I programmi C si memorizzano in file con estensione `.c` in Unix.

Supponendo quindi di aver salvato il programma precedente nel file `ciao_mondo.c`, per compilarlo usiamo il comando `cc` (C compiler) o `gcc` in Linux (GNU C compiler) nel modo seguente:

```
> gcc ciao_mondo.c
```

Il risultato del precedente comando è un file **binario** `a.out` che contiene l'immagine dell'**eseguibile** da caricare in memoria. Quindi digitando

```
> ./a.out
```

verrà stampata sullo standard output la stringa `ciao, mondo!` seguita da un newline.

Per ottenere un file eseguibile con un nome più significativo di `a.out`, è sufficiente specificarlo con l'opzione `-o`:

```
> gcc -o ciao_mondo ciao_mondo.c
> ./ciao_mondo
```

Un esempio di programma C

```
#include <stdio.h>

/* il programma stampa la tabella Fahrenheit-Celsius
   per l'intervallo di valori Fahrenheit da 0 a 300 */

main()
{
    float fahr, celsius;  /* dichiarazione di 2 variabili di tipo float */
    int lower, upper, step; /* dichiarazione di 3 variabili di tipo int */

    lower=0; upper=300; step=20; /* inizializzazione variabili */
    printf("Tabella Fahrenheit-Celsius\n");
    fahr=lower;

    while(fahr <= upper) /* ciclo while */
    {
        celsius = (5.0/9.0)*(fahr-32.0);
        printf("%3.0f %6.1f\n",fahr,celsius);
        fahr=fahr+step;
    }
}
```

La funzione `printf` e le sequenze di escape

Il comando

```
printf("%3.0f %6.1f\n",fahr,celsius);
```

prende come primo argomento una stringa di caratteri da stampare (e.g., `"%3.0f %6.1f\n"`) in cui ogni occorrenza del simbolo `%` indica il punto in cui devono essere sostituiti, nell'ordine, il 2^o, 3^o, ... argomento.
I caratteri successivi ad ogni `%` indicano il formato in cui deve essere stampato l'argomento.

Ad esempio, `%3.0f` indica che l'argomento deve essere di tipo `float` (a virgola mobile) e che devono essere stampati almeno 3 caratteri per la parte intera e nessun carattere per la parte decimale.

Alcune sequenze di escape comunemente usate per stampare **caratteri speciali** nella stringa fornita come primo argomento a `printf` sono:

`\n` : newline	`\b` : backspace
`\t` : tab	`\"` : doppi apici
`\\` : backslash	

Un altro programma per la conversione Fahrenheit-Celsius

```
#include <stdio.h>

#define LOWER 0
#define UPPER 300
#define STEP 20

main()
{
    float fahr;

    for(fahr=LOWER; fahr<=UPPER; fahr=fahr+STEP)
        printf("%3.0f  %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

Una direttiva al preprocessore della forma
`#define` *nome valore*
definisce una **costante simbolica** *nome* che, al momento della precompilazione, verrà rimpiazzata in tutto il programma (purché non campaia all'interno di apici o faccia parte di un altro identificatore) dalla **sequenza di caratteri** *valore*. Si noti quindi che le costanti simboliche **non sono** variabili; infatti per distinguerle vengono convenzionalmente scritte in maiuscolo.

I tipi base del C

`int`	interi
`float`	floating-point a precisione singola
`char`	caratteri (un singolo byte)
`short` (`short int`)	intero corto
`long` (`long int`)	intero lungo
`double`	floating-point a precisione doppia

In C esistono due tipi di **conversioni di tipo**:

1. **Promozioni**: conversioni automatiche

   `char` $\rightsquigarrow$ `short` $\rightsquigarrow$ `int` $\rightsquigarrow$ `long` $\rightsquigarrow$ `float` $\rightsquigarrow$ `double`

2. **Cast**: conversione esplicita (nel verso opposto); per esempio: `x=(int)5.0;`

Esempio I: I/O di caratteri

I seguenti programmi leggono i caratteri dallo standard input e li stampano sullo standard output, fintanto che non viene letto il carattere speciale di End Of File:

```
#include <stdio.h>

main()
{
    int c;

    c=getchar();

    while(c != EOF)
    {
        putchar(c);
        c = getchar();
    }
}
```

Versione "compatta":

```
#include <stdio.h>

main()
{
    int c;

    while((c = getchar()) != EOF)
    {
        putchar(c);
    }
}
```

Esempio II: conteggio di caratteri

I seguenti programmi implementano la funzionalità del comando Unix `wc -c`:

```
#include <stdio.h>


main()
{
    long nc;

    nc = 0;

    while(getchar() != EOF)
    {
        ++nc;
    }

    printf("%ld\n",nc);
}
```

```
#include <stdio.h>

main()
{
    long nc;

    for(nc = 0; getchar() != EOF; ++nc);

    printf("%ld\n",nc);
}
```

Ipotizzando di salvare uno dei due programmi nel file `contachar.c`, compilando con `gcc -o contachar contachar.c` si ottiene un eseguibile tale che il comando `./contachar < file` è equivalente al comando `wc -c file`.

Conteggio di linee

Il seguente programma implementa la funzionalità del comando Unix `wc -l`:

```
#include <stdio.h>

main()
{
    int c, nl;

    nl = 0;

    while((c = getchar()) != EOF)
        if (c == '\n')
            ++nl;

    printf("%d\n",nl);
}
```

Si noti che in C un carattere tra apici è un valore intero che corrisponde al valore numerico del carattere nel set di caratteri della macchina (e.g., `'A'` è il valore 65 in ASCII).

Esercizi

- Scrivere un programma C che stampi il valore della costante simbolica `EOF`.

- Scrivere un programma C che conti il numero di spazi, tab e newline (*whitespace characters*) presenti nei caratteri immessi sullo standard input.

- Scrivere un programma C che stampi un istogramma orizzontale (utilizzando il carattere -) raffigurante le lunghezze delle parole immesse sullo standard input (si considerino come delimitatori di parola i *whitespace characters*).

- Scrivere un programma C che conti il numero di parole immesse sullo standard input, sapendo che l'operatore logico or si denota con i caratteri || (si considerino come delimitatori di parola i *whitespace characters*).

Inizializzazione di vettori e puntatori a caratteri

- È possibile inizializzare un vettore (oltre che con un ciclo che assegni un valore ad ogni singolo elemento) direttamente in un colpo solo:

  ```
  int v[4]={10, 20, 30, 40};
  ```

- È anche possibile inizializzare un array in fase di dichiarazione senza specificare esplicitamente il numero di elementi:

  ```
  char string[]="ciao, mondo";
  ```

  In questo caso, verrà automaticamente allocato lo spazio necessario per contenere i caratteri della stringa `ciao, mondo` più il carattere terminatore `\0`.

- Nel caso di un puntatore a caratteri possiamo utilizzare la seguente dichiarazione

  ```
  char *pstring="ciao, mondo";
  ```

- Un esempio più complesso:

  ```
  char *line[]={"abc", "def", "ghi"};
  ```

Argomenti sulla linea di comando

Analogamente a quanto succede con i comandi Unix, è possibile scrivere dei programmi C che accettano argomenti sulla linea di comando:

```
#include <stdio.h>

main(int argc, char *argv[]) /* argc: argument count (n. argomenti+1) */
                             /* argv: argument vector (puntatore ad   */
                             /* un vettore di stringhe che contiene   */
                             /* gli argomenti).                       */
{
    int i;

    for(i=1; i<argc; i++)
        printf("%s\n",argv[i]);

    printf("\n");
    return 0;
}
```

Il programma precedente emula il comando Unix `echo`, nel senso che accetta un numero arbitrario di comandi e li stampa sullo standard output.

N.B.: `argv[0]` è il nome del programma C compilato, mentre `argv[argc]` è la costante `0` (i.e., un **null pointer**).

Input formattato: `scanf` e `sscanf` (I)

- La funzione di libreria
  `int scanf(char *format,...);`
  legge i caratteri dallo standard input interpretandoli secondo il formato specificato dal primo argomento e memorizzando i risultati nei rimanenti argomenti, che **devono** essere dei **puntatori**.
  Ad esempio `scanf("%f %d", &x, &i);` legge dallo standard input un numero a virgola mobile ed un intero e li assegna, rispettivamente, alle variabili `x` e `i` (si noti l'uso dell'operatore `&` per ottenere gli indirizzi delle variabili argomento).

- L'esecuzione di `scanf` termina quando esaurisce l'argomento `format`, quando l'input non soddisfa le specifiche od in caso di end-of-file.

- Il valore restituito da `scanf` rappresenta il numero di elementi in input che sono stati memorizzati con successo negli argomenti corrispondenti. In caso di end-of-file viene invece restituito il valore `EOF`.

- La funzione
  `int sscanf(char *string, char *format,...);`
  si comporta esattamente come `scanf` tranne per il fatto di leggere i caratteri dalla stringa puntata da `string` invece che dallo standard input.

Input formattato: `scanf` e `sscanf` (II)

- La stringa di formato può essere costituita dai seguenti elementi:
  - spazi o tabulazioni (vengono ignorati);
  - caratteri ordinari (diversi da `%`) che dovranno poi corrispondere esattamente ai caratteri in input diversi dai **white space characters**;
  - **specifiche di conversione** che iniziano con il carattere `%` seguito da un "suppression character" `*` (opzionale), da un numero indicante la lunghezza massima del campo (opzionale), da un carattere fra `h`, `l`, `L` indicante la "grandezza" del valore (opzionale) e da un **carattere di conversione**:

`d`	intero decimale
`i`	intero (eventualmente ottale, se preceduto da uno `0`, o esadecimale, se preceduto da `0x` o `0X`).
`o, x`	intero ottale o esadecimale rispettivamente
`c`	carattere (eventualmente anche white space)
`s`	stringa di caratteri
`e, f, g`	numero a virgola mobile

- Normalmente gli input field (a meno che non si usi il carattere di conversione `c`) sono sequenze consecutive di caratteri **non** white space che si estendono fino al prossimo carattere white space o fino alla lunghezza massima di campo specificata.

Esempi d'uso di `scanf`

- Nel caso si vogliano leggere linee del tipo
  `5 feb 2003`
  si può utilizzare la `scanf` seguente:
  `scanf("%2d %3s %4d", &giorno, mese, &anno);`
  dove `giorno`, `mese` ed `anno` sono dichiarati come segue:

  ```
  int giorno, anno;
  char mese[4];
  ```

  Si noti che il vettore `mese` ha dimensione 4 per permettere la memorizzazione di nomi abbreviati di mesi dell'anno (3 caratteri) più il carattere nullo di terminazione.

- L'istruzione
  `scanf("%f %*f %f", &x, &y);`
  legge tre numeri in virgola mobile e memorizza il primo in `x` ed il terzo in `y`, saltando il secondo (si noti l'uso del "suppression character" `*`).

Le funzioni in C

I programmi C sono costituiti da **definizioni di variabili** e **funzioni**.

Una definizione di funzione ha il seguente formato:

```
tipo-ritornato nome-funzione(lista-parametri)
{
   dichiarazioni
   istruzioni
}
```

Le definizioni di funzioni possono comparire in qualsiasi ordine all'interno di uno o più **file sorgente**, ma non possono essere spezzate in più file.

Esempio

Il seguente programma è costituito da una funzione `power(m,n)` che eleva un intero m alla potenza intera n, e dalla funzione speciale `main` che utilizza `power`.

```
# include <stdio.h>
int power(int m, int n); /* prototipo */
main()
{
   int i;
   for  (i=0; i<10; ++i)
      printf("%d %d %d\n", i, power(2,i), power(-3,i));
   return 0;
}
int power(int base, int n)
{
   int i,p;
   p=1;
   for (i=1; i<=n; ++i)
      p=p*base;
   return p; /* restituisce il valore di p al chiamante */
}
```

Analisi dell'esempio

La dichiarazione `int power(int m, int n)` all'inizio del programma è detta **prototipo** della funzione, indica che `power` si aspetta due argomenti interi e restituisce un intero. Il prototipo deve essere in accordo con la definizione della funzione stessa (a parte per i nomi dei parametri).

La prima linea della funzione `power` dichiara i nomi e i tipi dei parametri e il tipo del risultato. I nomi dei parametri sono locali a `power`, così come le variabili i, p dichiarate all'interno di `power`.

L'istruzione `return` ritorna il controllo al chiamante, eventualmente restuituendo il valore specificato.
Per le funzioni in cui l'istruzione `return` non compare, il controllo ritorna al chiamante alla fine dell'esecuzione della funzione.
Anche la funzione speciale `main` può avere un'istruzione `return`, che ritorna il controllo al chiamante, cioè l'ambiente in cui il programma è eseguito.

Argomenti: chiamata per valore e per riferimento

In C tutti gli argomenti delle funzioni, che **non** sono **vettori** nè **puntatori**, sono passati **per valore**. Cioè le funzioni lavorano su copie dei parametri e non modificano i parametri passati dal chiamante.

Al contrario, argomenti **vettore** o **puntatore** sono passati **per riferimento**, cioè viene passato l'**indirizzo** dei parametri e la funzione lavora sui parametri originali.

Variabili e scope

Le variabili dichiarate all'**interno** delle funzioni sono **locali** alle funzioni e sono dette **automatiche**, in quanto sono create al momento della chiamata della funzione e cessano di esistere quando questa termina.
Perciò una variabile automatica deve essere sempre inizializzata ad ogni chiamata, altrimenti può provocare errore.

Le variabili dichiarate come `static` sono variabili che conservano il loro valore tra una chiamata e l'altra di una funzione.

Le variabili **esterne** (globali) sono **definite al di fuori** delle funzioni. Tali variabili devono essere anche **dichiarate all'interno** di ogni funzione che le usa come variabili `extern`. Nel caso in cui le variabili globali siano dichiarate nello stesso file sorgente della funzione che ne fa uso, la loro ridefinizione con la parola chiave `extern` all'interno di quest'ultima non è necessaria.

Le variabili dichiarate come `register` sono collocate in **registri** della macchina e quindi permettono un accesso più rapido. Ci sono però delle limitazioni (e.g. non si può accedere all'indirizzo di una variabile register); inoltre il compilatore può ignorare la dichiarazione `register`.

Esempio

Programma che legge un insieme di linee di testo (le stringhe di caratteri si rappresentano in C come vettori di caratteri; l'ultimo carattere della stringa è il carattere \0 che funge da **terminatore**) e stampa la più lunga.

```
#include <stdio.h>
#define MAXLINE 1000  /* lunghezza massima di una linea */
int getline(char line[], int maxline);
void copy(char to[], char from[]);
main()
{
   int len;  /* lunghezza della linea corrente */
   int max;  /* massima lunghezza trovata finora */
   char line[MAXLINE];  /* linea di input corrente */
   char longest[MAXLINE];  /* linea piu' lunga corrente */
   max=0;
   while ((len=getline(line, MAXLINE)) > 0)
      if (len > max)
      {  max=len;
         copy(longest, line);
      }
```

. . . esempio

```
    if (max > 0)  /* c'era almeno una linea in input */
        printf("%s\n", longest);
    return 0;
}
```

Esercizio: definire le funzioni

```
/* getline: legge e carica in s una linea, ritorna la lunghezza */
int getline(char s[], int lim)

/* copy: copia from in to; assume che to sia sufficientemente ampio */
void copy(char to[], char from[])
```

Versione alternativa

Versione alternativa del programma per la stampa della linea più lunga, che definisce le variabili `line`, `longest` e `max` come variabili esterne.

```
#include <stdio.h>
#define MAXLINE 1000  /* lunghezza massima di una linea */

int max;  /* massima lunghezza trovata finore */
char line[MAXLINE];  /* linea di input corrente */
char longest[MAXLINE];  /* linea piu' lunga corrente */

int getline(void);
void copy(void);

main()
{
   int len;
   extern int max;
   extern char longest[MAXLINE];

   max=0;
```

...

```
   while ((len=getline()) > 0)
      if (len > max)
      {  max=len;
         copy();
      }
   if (max > 0)  /* c'era almeno una linea in input */
      printf("%s\n", longest);
   return 0;
}
```

Esercizio: definire le funzioni

```
int getline(void)

void copy(void)
```

Esercizi

Scrivete le seguenti funzioni:

1. `reverse(s)`, che inverte la stringa di caratteri s. Usate tale funzione per scrivere un programma che inverte le linee di un testo in input.

2. `strindex(s,t)` che restituisce la posizione, cioè l'indice di inizio della stringa t nel vettore s, oppure -1 se t non compare in s.

3. `strrindex(s,t)` che restituisce la posizione dell'occorrenza più a destra di t in s, oppure -1 se t non compare in s.