

# I Progetto: La shell Unix (I)

Progettare uno script per la shell Bash che prenda un parametro sulla linea di comando. Il parametro deve essere una directory a cui l'utente possa accedere, leggendone il contenuto e memorizzandovi nuovi file. Se la directory non esiste, deve essere creata.

Lo script dovrà archiviare (copiandoli) nella directory suddetta i file presenti nella propria home e nelle relative sottodirectory **non modificati** da più di una settimana.

Prevedere la possibilità di invocare lo script con l'opzione `-z` che, invece di copiare i file che rispondono al requisito sopra esposto, crea un unico archivio `.tar.gz` (creato con il comando `tar` e contenente i file), memorizzando quest'ultimo nella directory passata come primo parametro.

- Si gestiscano opportunamente gli errori.
- Si commenti opportunamente il codice.
- Si provi lo script su alcuni esempi significativi.

# I Progetto: La shell Unix (II)

Per quanto riguarda la copia dei file nella cartella specificata come parametro, si può presentare il problema per cui vi siano più file con lo stesso nome (presenti in parti diverse del file system) che rispondano ai requisiti richiesti (ovvero, non siano stati modificati da più di una settimana). In questo caso sono possibili varie strategie di implementazione, fra cui:

- copiare il primo file e tralasciare le copie successive con lo stesso nome, avvertendo l'utente (magari inserendo in un file di log i percorsi dei file non copiati);
- consentire all'utente di decidere se sovrascrivere l'ultimo file copiato con la nuova copia avente lo stesso nome (questo per ogni occorrenza di file con lo stesso nome);
- replicare la struttura delle directory nella cartella di backup (eliminando così il problema alla radice);
- aggiungere un suffisso con un numero progressivo ad ogni copia di file con lo stesso nome, e.g., `file_1`, `file_2`, `file_3` ecc.

# 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 esiste una particolare, chiamata `main`, da cui **inizia l'esecuzione** e che quindi deve essere presente in ogni programma. Le due parentesi () 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<sup>o</sup>, 3<sup>o</sup>, ... 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:

<code>\n</code> : newline	<code>\b</code> : backspace
<code>\t</code> : tab	<code>\"</code> : doppi apici
<code>\\</code> : 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 compaia 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

<code>int</code>	interi
<code>float</code>	floating-point a precisione singola
<code>char</code>	caratteri (un singolo byte)
<code>short (short int)</code>	intero corto
<code>long (long int)</code>	intero lungo
<code>double</code>	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*).