Corso di Laboratorio di Sistemi Operativi
A.A. 2016–2017

Lezione 7

Ivan Scagnetto
ivan.scagnetto@uniud.it
Nicola Gigante
gigante.nicola@spes.uniud.it

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine

A.A. 2016–2017 - Primo Semestre - 26/10/2016

Informazioni di servizio

Info su questa parte del corso:

- Argomenti:
  - Linguaggio C
  - Programmazione di sistema in ambiente Unix
- E-mail: `gigante.nicola@spes.uniud.it`
- Materiale aggiuntivo per il corso:
  `http://users.dimi.uniud.it/~nicola.gigante/teaching`
- Domande? Per accordarsi basta un'email.
- Ufficio: Nodo Nord n. 2.
  All'entrata della biblioteca, salire le scale fino al secondo piano, prima porta a sinistra della finestra.

Introduzione

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.

Perchè imparare il C

- Linguaggi derivati dal C: C++, Objective C, C♯, ...
- Linguaggi con sintassi ispirata al C: Java, JavaScript, Perl, PHP, C shell di UNIX/Linux, ...
- Qualunque linguaggio deve fare i conti con l'hardware: avere una buona conoscenza del C aiuta a capire cosa sta succedendo dietro le quinte.
- Linguaggio cardine dei sistemi UNIX:
  - Le chiamate di sistema sono definite come funzioni C
  - Qualsiasi linguaggio quindi alla fine passa per il C per interfacciarsi con il sistema operativo
  - Questo vale anche su Windows
- Spesso è necessario scrivere in C parti delicate di software scritto in altri linguaggi meno efficienti (Python, Java, ecc...)

Utilizzi del linguaggio C (e C++)

- Programmazione di sistema, sistemi embedded.
- Software numerico/scientifico: MATLAB, NumPy, SciKit, ...
- Implementazione di compilatori, interpreti, librerie per altri linguaggi di programmazione (e.g., Python, Perl, PHP).
- *Lingua franca* per la comunicazione tra linguaggi diversi
- 3D graphics APIs: OpenGL, DirectX/Direct3D
- 3D game engines (in C++, in realtà): Ogre3D, Unreal Engine, NetImmerse Engine, Unity, Bullet, PhysX, ...
- Computazione su GPU: OpenCL, CUDA
- Computer Vision e elaborazione dati audio/video, es. OpenCV, OpenAL
- Qualunque ambito in cui siano richieste prestazioni, efficienza nell'utilizzo delle risorse e portabilità del codice.

Lo Standard ISO e le sue implementazioni

Il linguaggio C è definito da uno **standard internazionale ISO**.

- Tre versioni sono state ratificate finora: ISO C89, **C99** e C11.
- Ogni versione aggiunge qualche funzionalità lasciando completa compatibilità all'indietro.
- Il produttore dell'hardware e/o del sistema operativo fornisce il proprio **compilatore** e/o la propria implementazione della **libreria standard**.
  - Linux: gcc, clang, icc, ...
  - Windows: MSVC, icc, gcc, ...
  - Mac OS X: clang, gcc, icc, ...
- Esiste quindi un linguaggio ma molte implementazioni diverse.
- Attenzione a non confondere il **compilatore** con **l'ambiente di sviluppo**, es. MSVC vs. Visual Studio, clang vs Xcode, ...

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:

```c
#include <stdio.h>

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

  return 0;
}
```

- 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.

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:

```c
#include <stdio.h>

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

  return 0;
}
```

- Ogni programma C è composto da **dichiarazioni** di variabili, tipi, e funzioni (contenenti istruzioni).
- fra queste ne esiste 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 parola chiave `int` indica che `main` **restituisce** un numero intero.

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:

```c
#include <stdio.h>

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

  return 0;
}
```

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

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:

```c
#include <stdio.h>

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

  return 0;
}
```

- L'istruzione `return 0;` restituisce il numero zero, e termina l'esecuzione della funzione (e quindi del programma).
- Zero significa che il programma è andato a buon fine, come per convenzione nel mondo Unix.

Compilazione di programmi C

In ambiente Linux e Mac OS X esistono due compilatori C **open source** e liberamente disponibili: GCC (GNU C Compiler) e Clang.

- Funzionano entrambi su Linux, Mac OS X, Windows, iOS, Android, quasi tutti gli UNIX e tante piattaforme embedded.
- Supportano entrambi tutte le versioni di ISO C (e C++).
- GCC è il compilatore di "default" su sistemi Linux (e Android).
- Clang è il compilatore ufficiale su Mac OS X (e iOS).

Compilazione di programmi C

In ambiente Linux e Mac OS X esistono due compilatori C **open source** e liberamente disponibili: GCC (GNU C Compiler) e Clang.

- L'uso di uno o dell'altro è equivalente:
  - Piccole differenze nell'interpretazione degli standard
  - Opzioni da riga di comando uguali o molto simili
  - Nella maggioranza dei casi completamente intercambiabili
- I nostri esempi useranno Clang perchè:
  - I messaggi di errore sono più chiari e comprensibili
  - Compila di default in modalità C11

Compilazione di programmi C

Invocare il compilatore

Per compilare un programma, dopo averlo salvato in un file, ad esempio `programma.c`, si invoca il compilatore:

```
$ clang programma.c
```

Se il programma è sintatticamente corretto, il compilatore produrrà un file **eseguibile** chiamato `a.out`, che può essere eseguito:

```
$ ./a.out
Ciao mondo!
```

Compilazione di programmi C

Invocare il compilatore

L'opzione `-o` permette di dare un nome diverso al file di output:

```
$ clang programma.c -o programma
$ ./programma
Ciao mondo!
```

I tipi base del C

Tipo	Significato
`char`	carattere (un singolo byte)
`short int`	intero corto
`int`	intero
`long int`	intero lungo
`float`	floating-point a precisione singola
`double`	floating-point a precisione doppia
`unsigned int`	intero senza segno (anche `long` o `short`)

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;
   ```

Sintassi di base

La sintassi di base è molto familiare per chi conosce già, ad esempio, il Java:

- Dichiarazione di variabili (`int x = 0;`)
- Costrutti di controllo e cicli (`if`, `for`, `while`)
- Espressioni e operatori:
  - Operatori aritmetici (`+`, `-`, `*`, `/`, `%`, ecc...)
  - Operatori logici (`&&`, `||`, `!`, ecc...)
  - Precedenza degli operatori
- Costanti numeriche (`42`, `3.14`), stringhe (`"Hello world"`), caratteri (`'A'`), ecc...
- **Attenzione**: Le somiglianze si limitano alla sintassi di base. In tutto il resto i due linguaggi sono profondamente diversi.

Espressioni booleane

La prima differenza importante rispetto al C++ e al Java è che in C non esiste* il tipo `bool`.

- Il costrutto `if`, e i cicli `for` e `while`, si aspettano delle espressioni di tipo intero nelle proprie condizioni di controllo.
- Il valore **zero** viene interpretato come **falso**.
- Qualsiasi valore diverso da zero viene interpretato come **vero**.

```c
#include <stdio.h>

int main() {

  int x = 0;

  if(x)
    printf("x e' falso\n");
  else
    printf("x e' vero\n");

  return 0;
}
```

Un esempio di programma C

Il seguente programma stampa la tabella Fahrenheit-Celsius per l'intervallo di valori Fahrenheit da 0 a 300:

```c
#include <stdio.h> // Funzioni di I/O dalla libreria standard

int main()
{
  float fahr = 0; // dichiarazione di una variabile di tipo float

  printf("Tabella Fahrenheit-Celsius:\n");

  while(fahr <= 300) // ciclo while
  {
      float celsius = (5.0 / 9.0) * (fahr - 32.0);
      printf("%3.0f °F -> %6.1f °C\n", fahr, celsius);
      fahr = fahr + 20;
  }

  return 0;
}
```

La funzione `printf`

L'istruzione

```
printf("%3.0f °F -> %6.1f °C\n", fahr, celsius);
```

chiama la funzione `printf`, comunemente usata per stampare messaggi sullo standard output nei programmi C.

- Il primo argomento è una stringa di caratteri da stampare (`"%3.0f °F -> %6.1f °C\n"`) in cui ogni occorrenza del simbolo `%` indica il punto in cui devono essere sostituiti, nell'ordine, il 2°, 3°, ... argomento
- I caratteri successivi ad ogni `%` indicano il **tipo** dell'argomento e il formato in cui deve essere stampato.
- Ad esempio, `%3.0f` indica che l'argomento deve essere di tipo `float` e che devono essere stampati almeno 3 caratteri e nessun carattere per la parte decimale.

Sequenze di escape

All'interno delle stringhe, le **sequenze di escape** permettono di specificare caratteri particolari:

Sequenza	Significato
`\n`	Newline (a capo)
`\t`	Tab
`\\`	Singola backslash `'\'`
`\"`	Doppi apici

Attenzione

Non vanno confuse le sequenze di escape, che fanno parte della sintassi del **linguaggio**, con gli specificatori di formato di **printf**, che invece fanno parte del modo in cui una singola funzione interpreta il proprio input.

Il preprocessore

Ogni file C, prima della compilazione, viene **preprocessato**.

- Il preprocessore trasforma il codice interpretando delle direttive.
- Le direttive sono righe di codice che cominciano con un cancelletto #, seguito dal nome della direttiva vera e propria
- Il compilatore vede solo il risultato del preprocessing.

Alcune utili direttive del preprocessore

```
#include <nomefile.h>
```

Include testualmente il file `nomefile.h`. Usato per includere i **file di intestazione** della libreria standard.

```
#define NOME valore
```

Definisce **costanti simboliche**.

- Dopo tale direttiva, il preprocessore sostituirà testualmente ogni occorrenza di `NOME` con `valore`.
- La sostituzione non avviene all'interno di stringhe o all'interno di altri identificatori.
- Le costanti simboliche **non sono** variabili; infatti per distinguerle vengono convenzionalmente scritte in maiuscolo.

Un altro programma per la conversione Fahrenheit-Celsius

```c
#include <stdio.h>

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

int main()
{
  for(float fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
    printf("%3.0f °F -> %6.1f °C\n", fahr, (5.0/9.0) * (fahr - 32));

  return 0;
}
```

Esempi

Esempio I

I/O di caratteri

I seguenti programmi implementano una versione semplificata del comando `cat`: leggono caratteri dallo standard input e li ripetono tali e quali sullo standard output, finchè l'input non termina (EOF, ovvero End Of File).

Con ciclo `while`:

```
#include <stdio.h>

int main()
{
  int c = getchar();

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

  return 0;
}
```

Con ciclo `for`:

```
#include <stdio.h>

int main()
{
  for(int c = getchar(); c != EOF; c = getchar())
  {
      putchar(c);
  }

  return 0;
}
```

Nota: la shortcut da tastiera per immettere il carattere speciale di EOF sul terminale è `Ctrl-D`.

Esempio II

Conteggio di caratteri

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

Con ciclo `while`:

```
#include <stdio.h>

int main()
{
  long nc = 0;

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

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

  return 0;
}
```

Con ciclo `for`:

```
#include <stdio.h>

int main()
{
  long nc = 0;
  for(; getchar() != EOF; ++nc);

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

  return 0;
}
```

Esempio III

Conteggio di linee

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

```
#include <stdio.h>

int main()
{
  long nc = 0;

  for(int c = getchar(); c != EOF; c = getchar())
    if(c == '\n')
      ++nc;

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

  return 0;
}
```

Esercizi

Esercizi

Per mercoledì 2 novembre

1. Scrivere un programma C che stampi il valore della costante simbolica `EOF`.
2. Scrivere un programma C che conti il numero di spazi, tab e newline (*whitespace characters*) presenti nei caratteri immessi sullo standard input.
3. Scrivere un programma C che stampi un istogramma orizzontale (utilizzando il carattere `'-'`) raffigurante le lunghezze delle parole (delimitate da *whitespace characters*) immesse sullo standard input (parola per parola).
4. Scrivere un programma C che conti il numero di parole immesse sullo standard input (si considerino come delimitatori di parola i *whitespace characters*).

Soluzioni

Esercizio 1

```c
#include <stdio.h>

int main() {

  printf("Valore numerico della costante EOF: %d\n", EOF);

  return 0;
}
```

Soluzioni

Esercizio 2

```c
#include <stdio.h>

int main() {
  int tab = 0, spazi = 0, newline = 0;

  for(int c = getchar(); c != EOF; c = getchar()) {
    switch(c) {
      case ' ':
        spazi++;
        break;
      case '\t':
        tab++;
        break;
      case '\n':
        newline++;
        break;
    }
  }

  printf("Spazi: %d\nTabulazioni: %d\nNewline: %d\n", spazi, tab, newline);

  return 0;
}
```

Soluzioni

Esercizio 3

```c
#include <stdio.h>

int main() {
  int i = 0, n = 0;

  for(int c = getchar(); c != EOF; c = getchar())
  {
    if(c != ' ' && c != '\t' && c != '\n') {
      n++;
    } else {

      for(i = 0; i < n; i++)
        printf("-");

      if(n > 0)
        printf("\n");

      n = 0;
    }
  }

  return 0;
}
```

Soluzioni

Esercizio 4

```c
#include <stdio.h>

int main() {
  int n = 0, inizio_parola = 0;

  for(int c = getchar(); c != EOF; c = getchar()) {

    if(c ==' ' || c == '\t' || c == '\n')
    {
      if(inizio_parola) {
        n++;
        inizio_parola=0;
      }
    } else
      inizio_parola=1;
  }

  if(inizio_parola)
    n++;

  printf("Numero di parole: %d\n",n);
}
```