Lezione 6: Introduzione alla programmazione in C

Laboratorio di Elementi di Architettura e Sistemi Operativi

26 Marzo 2013

Introduzione al C

C vs. Java

- Java: linguaggio ad oggetti
- C: linguaggio procedurale
  - No classi, no oggetti, no costruttori, ...
  - Separazione tra dati e codice
- Diverso flusso di esecuzione
- Sintassi simile
  - Stesse istruzioni per assegnamenti e controllo del flusso
  - Diverse istruzioni di I/O
- *Differenza fondamentale*
  - A oggetti vs. procedurale!

Java

```
public class xyz
{
  <dichiarazione di attributi>
  <dichiarazione di metodi>
}
```

C

```
<dichiarazione di attributi>
<dichiarazione di funzioni>
```

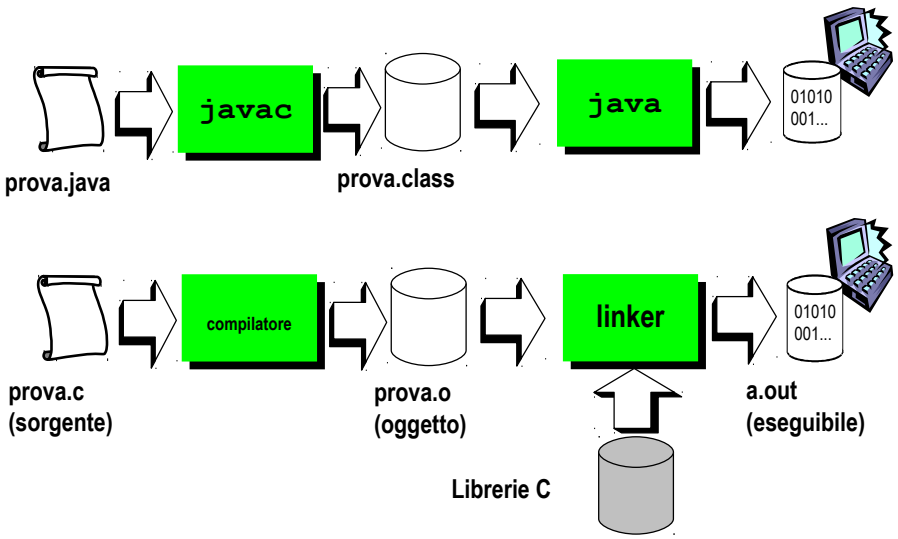
HelloWorld.java

```
public class HelloWorld
{
  public static void main(String args[])
  {
    int i;
    System.out.println("Hello World!");
  }
}
```

HelloWorld.c

```
void main(int argc, char *argv[])
{
  int i;
  printf("Hello World!\n");
}
```

Il flusso di esecuzione



- Apparente similitudine
- In realtà molto diverso:
  - Esecuzione Java = compilatore + interprete (JVM)
  - Esecuzione C = compilatore + linker
  - Il Linker svolge due funzioni essenziali:
    - Collegamento a librerie di codice precedentemente scritto
    - Binding degli indirizzi simbolici in indirizzi rilocabili
  - In ambiente UNIX/Linux, compilazione + link realizzati da un singolo programma (compilatore C): `gcc`

Il compilatore C

- `gcc`
  - GNU C Compiler
  - Non è un comando UNIX!
- Uso di base:
  - `gcc nomefile.c` genera un file eseguibile `a.out`
- Opzioni (combinabili tra loro):
  - `gcc -g`: genera le informazioni per il *debugging*
  - `gcc -o nomefile`: genera un eseguibile con il nome `nomefile`
  - `gcc -c`: genera solo il file `.o` senza fare linking
  - `gcc -I directory`: cerca i file da compilare anche in `directory`
  - `gcc -lnome`: fai il link con la libreria `libnome.a`
- Esempi:
  - `gcc prova.c` Genera un eseguibile con il nome `a.out`
  - `gcc -o prova.x prova.c` Genera un eseguibile con il nome `prova.x`
  - `gcc -o prova.x -g prova.c` Genera `prova.x` con info di debugging
  - `gcc -c prova.c` Genera il file `prova.o`, cioè non effettua il linking
  - `gcc -o prova.g -g -lm prova.c` Genera un eseguibile con il nome `prova.g`, info di debugging e usando la libreria `libm.a`.

Struttura di un programma C

Versione minimale

```
Dichiarazioni globali

main()
{
  Dichiarazioni locali

  Istruzioni
}
```

Versione generale

```
Dichiarazioni globali

main()
{
  Dichiarazioni locali
  Istruzioni
}

funzione1()
{
  Dichiarazioni locali
  Istruzioni
}
...
funzioneN()
{
  Dichiarazioni locali
  Istruzioni
}
```

- Dichiarazioni globali
  - Elenco dei dati usati in tutto il programma e delle loro caratteristiche (tipo)

- Dichiarazioni locali
  - Elenco dei dati usati dalle singole funzioni con il relativo tipo
  - Il `main` è una funzione come le altre anche se ha un nome speciale

Preprocessore C

- La prima fase della compilazione (trasparente all'utente) consiste nell'invocazione del *preprocessore*
- La prima parte di un programma C contiene specifiche direttive per il preprocessore:
  - Inclusioni di file di definizioni (*header file*)
  - Definizioni di costanti
  - Altre direttive
- Individuate dal simbolo `#`

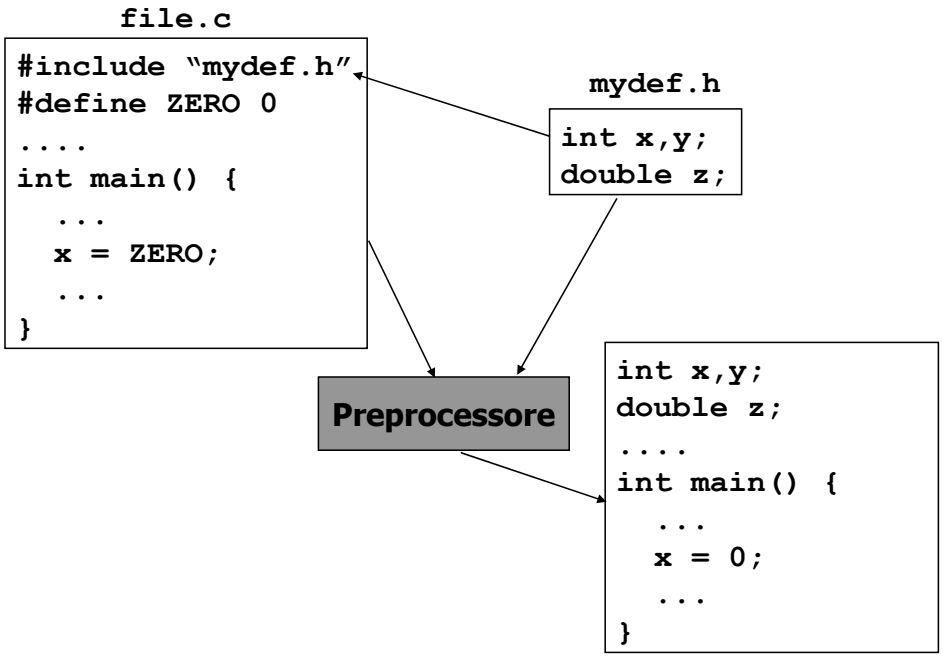
Direttive del preprocessore

`#include`

- Inclusione di un header file (tipicamente con estensione `.h`)
- Esempi
  - `#include <stdio.h>` *dalle directory di sistema*
  - `#include "myheader.h"` *dalla directory corrente*

`#define`

- Definizione di un valore costante
- Ogni riferimento alla costante viene espanso dal preprocessore al valore corrispondente
- È buona consuetudine mettere il nome TUTTO MAIUSCOLO
- Esempi
  - `#define FALSE 0`
  - `#define SEPARATOR "--------------"`



Le variabili

Definizione delle variabili

- Tutte le variabili devono essere definite prima di essere usate
- La definizione di una variabile:
  - riserva spazio in memoria
  - assegna un nome
- Richiede l'indicazione di:
  - tipo
  - modalità di accesso (variabili/costanti)
  - nome (identificatore)

Tipi di base (primitivi)

- Sono quelli forniti direttamente dal C
- Identificati da parole chiave:

`char`	caratteri ASCII ('a', 'b', '1')
`int`	interi (complemento a 2)
`float`	reali (floating point singola precisione)
`double`	reali (floating point doppia precisione)

- La dimensione precisa di questi tipi dipende dall'architettura (non è definita dal linguaggio C)

- Eccezione: `char` sempre su un byte (8 bit)
- I tipi di base possono avere dei *modificatori*, identificati da parole chiave da premettere al tipo:
  - `signed / unsigned`: applicabili ai tipi `char` e `int`
    - `signed`: valore numerico con segno
    - `unsigned`: valore numerico senza segno
  - `short / long`: applicabili al tipo `int`
    - 16 bit (short) / 32 bit (long)
    - Utilizzabili anche senza specificare `int`

Definizione di variabili

- Definizione singola: `tipo variabile;`
- Definizione multiple: `tipo variabile1, variabile2, ... , variabileN;`
- Esempi:

```
int x;
char ch;
long int x1,x2,x3;
double pi;
short int stipendio;
long y,z;
```

Usiamo nomi significativi!

```
int x0a11;      // NO!
int risultato;  // OK!
```

Valori

- Valori che rappresentano quantità fisse:

```
char
'a'
97

int, short, long
26          /* notazione decimale */
0x1a, 0X1A  /* notazione esadecimale */
032         /* notazione ottale (base 8) */
26L         /* costante forzata a long */
26U         /* costante forzata a unsigned */
26UL        /* costante forzata a unsigned long */

float, double
-212.6
-2.126e2 -2.126E2
-2.126e-2
```

Valori speciali

- Rappresentano caratteri ASCII non stampabili e/o "speciali"
- Ottenibili tramite "sequenze di escape": `\<codice ASCII ottale su tre cifre>`
- Esempi

```
'\007'
'\013'
```

- Caratteri speciali: `'\b'` backspace `'\f'` form feed (pagina nuova) `'\n'` line feed (riga nuova) `'\t'` tab (tabulazione)

Definizione di costanti

- Sintassi: `const tipo nomecostante = valore;`
- Esempi:

```
const double pigreco =  3.14159;
const char  separatore = '$';
const float aliquota = 0.2;
```

- Preferibile rispetto all'uso di `#define`;
- Per convenzione gli identificatori delle costanti sono tipicamente in MAIUSCOLO: `const double PIGRECO = 3.14159;`

Stringhe

- Le stringhe sono rappresentate come sequenze di caratteri terminate dal carattere `NULL` (`'\0'`)
- *Non sono un tipo di base del C!*
- Costanti stringa: `"sequenza di caratteri"`
- Il carattere `NULL` è già inserito dal compilatore
- Esempi:

```
"Ciao!"
"prima riga\nseconda riga"
```

Visibilità delle variabili

- Ogni variabile è definita all'interno di un preciso ambiente di visibilità (*scope*)
- Variabili globali:
  - Definite all'esterno del `main()` e di tutte le altre funzioni
- Variabili locali:
  - Definite all'interno di un blocco `{ ... }`
  - Esempi di blocco:
    - funzione `main`
    - altre funzioni
    - blocchi all'interno di costrutti `if` e `cicli`

- `n,x`
  visibili in tutto il file
- `a,b,c,y`
  visibili in tutto il `main`
- `d,z`
  visibili nel blocco `if`

```
int n;
double x;
main() {
  int a,b,c;
  double y;
  if (a > b)
  {
    int d;
    double z;
    ...
  }
  ...
}
```

Le istruzioni

- Assegnamenti: come in Java `a=3;` `c=d;` ...
- Operatori: simili a quelli Java (non tutti) `+`, `-`, `*`, `/`, ...
- Operatori di confronto: come quelli Java `a == 1`, `b != c`, `f < 3.5`, ...
- Operatori logici: come quelli Java `&&` (and), `||` (or), `!` (not)
- Istruzioni di controllo del flusso: come quelli Java `if`, `switch`, `while`, `for`

Le funzioni di I/O

La funzione `printf`

```
#include <stdio.h>
printf(stringa, arg1, arg2, ... , argN);
```

- `stringa` determina il *formato di stampa* degli argomenti
- Può contenere
  - Caratteri (stampati come appaiono)
  - Direttive di formato nella forma `%<carattere>`:

```
%d   intero
%u   unsigned
%s   stringa
%c   carattere
%x   esadecimale
%o   ottale
%f   float
%g   double
```

- Se il formato di stampa non corrisponde al tipo dell'argomento il risultato potrebbe essere errato
- `arg1, ... , argN` quantità (o espressioni) che si vogliono stampare
  - Associate alle direttive di formato nello stesso ordine
- Esempi

```
int   x = 2;
float z = 0.5;
char  c = 'a';
```

	Output:
`printf("%d %f  %c\n",x,z,c);`	`2 0.5  a`
`printf("%f***%c***%d\n",z,c,x);`	`0.5***a***2`
`printf("%f***%d***%d\n",z,c,x);`	`0.5***97***2`

La funzione `scanf`

```
#include <stdio.h>
scanf(stringa, arg1, arg2, ..., argN);
```

- `stringa`: come per `printf`
- `arg1`, ..., `argN`: le variabili a cui si vogliono assegnare valori
- Esempio:

```
int x;
float z;
scanf("%d %f", &x, &z);
```

IMPORTANTE
I nomi delle variabili vanno preceduti dall'operatore `&` che indica l'indirizzo della variabile