TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA
Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Linguaggio C: Introduzione

Valeria Cardellini

Corso di Calcolatori Elettronici
A.A. 2019/20

Docente

- Valeria Cardellini
  - Professore associato nel SSD ING-INF/05 (Sistemi di Elaborazione delle Informazioni)
  - Dipartimento di Ingegneria Civile e Ingegneria Informatica
  - Tel: 06 72597388
  - http://www.ce.uniroma2.it/~valeria
- Email: cardellini@ing.uniroma2.it
  - Indicare [CE1920] nell'oggetto della mail

- Ricevimento:
  - Quando: martedì dalle 10:00 alle 11:00
  - Dove: ufficio D1-17, edificio Ingegneria dell'Informazione

Perché il linguaggio C?

- Linguaggio di programmazione di alto livello, ma più "vicino" al calcolatore
  - Usa concetti relativamente semplici e vicini al funzionamento dell'hardware dei calcolatori
  - Ruolo centrale dei puntatori
- Linguaggio di sistema
  - Principali sistemi operativi scritti in C
  - Elevato livello di integrazione con sistemi operativi Unix-like (Linux, OS X, …)
- Efficienza
  - Molte istruzioni C traducibili direttamente con una singola istruzione di linguaggio macchina
- Indipendente dall'hardware

Perché il linguaggio C?

- 3° nella classifica dei linguaggi di programmazione più popolari secondo IEEE https://bit.ly/2IvmlTI

Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0

Argomenti trattati

- Studieremo C ANSI/ISO
- Tipi di dato primitivi
- Strutture di controllo
- Funzioni
- Array
- Puntatori
- Caratteri e stringhe
- Input/output
- Struct e union
- Strutture dati in C
- C nei corsi successivi: programmazione di sistema e programmazione di rete

Materiale didattico

- Numerosi libri, tra cui:
  - H.M. Deitel, P.J. Deitel, "C Corso Completo di Programmazione", Apogeo.
  - D. Ritchies, B. Kernighan, "The C Programming Language", 2nd ed., 1988
    - ✓ Il famoso K&R
  - Possono andare bene anche altri libri
- Numerose dispense tra cui:
  - http://twiki.di.uniroma1.it/pub/Infogen/DispenseSulLinguaggioC/Dispense_C.pdf
- Il manuale GNU C
  - http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf

Ambiente di sviluppo

- Due alternative

1. Usare un editor di testo e il compilatore **gcc** con interfaccia a riga di comando
   - Opzione consigliata per la prova di laboratorio
   - Scelta dell'editor dipendente da sistema operativo e proprie preferenze
     - ✓ Linux: Nano, GNU Emacs, Gedit, Vim, Sublime Text, …
     - ✓ Windows: Notepad++, …
     - ✓ Mac: Sublime Text, …
2. Usare un ambiente di sviluppo integrato (IDE, Integrated Development Environment)
   - Code::Blocks, CodeLite, Eclipse, Visual Studio Code, NetBeans, …

Un po' di storia

- Ideato negli anni '70 presso Bell Labs da Dennis Ritchies
  - Evoluzione di due linguaggi precedenti: BCPL e B
- Usato da Dennis Ritchies e Brian Kernighan per scrivere il sistema operativo Unix

Photo: Alcatel-Lucent

The Strange Birth and Long Life of Unix, https://bit.ly/2y2itUl

Il nostro primo programma in C: Hello World

- Osserviamo alcune caratteristiche del C e differenze rispetto a Python
- Cosa notiamo?

```
#include <stdio.h>

int main(void) {
   printf("Hello World\n");
   return 0;
}
```


Caratteristiche del linguaggio C

- Linguaggio di programmazione di alto livello
- General-purpose
- Compilato
- Imperativo di natura procedurale

Linguaggi di programmazione

- Alto vs. basso livello
- Generale vs. specifico
- Interpretato vs. compilato
- Paradigmi di linguaggi di programmazione

Alto livello vs. basso livello

- Distinzione in base al livello di astrazione
- Linguaggio di programmazione di basso livello (e.g., Assember): l'insieme di istruzioni è molto semplice
- Linguaggio di programmazione di alto livello (e.g., Pyhton): insieme di primitive più ricco e complesso

Generale vs. specifico

- Distinzione in base alla gamma di applicazioni
- In un linguaggio di programmazione generale (o general-purpose), l'insieme di primitive supporta un'ampia gamma di applicazioni
- Un linguaggio di programmazione specifico mira a un insieme specifico (e ristretto) di applicazioni
  - Es: MATLAB (MATrix LABoratory) è un linguaggio di programmazione specificamente progettato per il calcolo numerico (operazioni su vettori e matrici)

Interpretato vs. compilato

- Distinzione secondo il modo in cui viene eseguito il codice sorgente
- Nei linguaggi interpretati (ad es., Python), il codice sorgente è eseguito direttamente in fase di esecuzione (dall'interprete)
  - L'interprete controlla il flusso del programma passando attraverso ognuna delle istruzioni
- Nelle linguaggi compilati (ad es. C), il codice sorgente viene tradotto in un codice oggetto (dal compilatore) prima dell'esecuzione

Interpretato vs. compilato: C vs. Python

- Una delle principali differenze tra C e Python riguarda l'esecuzione dei programmi scritti in questi due linguaggi
- Con i programmi C, in genere si usa un compilatore, che trasforma il codice C in istruzioni in linguaggio macchina, che vengono successivamente eseguite
  - Il compilatore non esegue il programma ma genera un file eseguibile!
- Al contrario, con Python in genere si utilizza un interprete, che esegue le istruzioni nel programma, traducendole di volta in volta in istruzioni in linguaggio macchina

Interpretato vs. compilato: C vs. Python

- Un interprete legge il programma scritto dall'utente e lo esegue direttamente
- Un compilatore genera un file contenente la traduzione del programma nel codice nativo della macchina
- Linguaggio compilato: implicazioni fondamentali sulla progettazione del linguaggio
  - In C il compilatore ha tutte le informazioni per tradurre il programma in codice C senza dover effettivamente eseguire il programma

Compilazione in C: GCC

- GCC (GNU C Compiler)
- GCC (GNU Compiler Collection)
- http://gcc.gnu.org
- Compilatore multipiattaforma
  - Sviluppato dalla Free Software Foundation
- Nato inizialmente per il linguaggio C, gestisce anche C++, Fortran, Ada e Go
- Genera codice per varie piattaforme
  - X86, X86-64, I-64, ARM, SPARC
- Adottato come compilatore principale per lo sviluppo di vari sistemi operativi, tra cui Unix BSD, Mac OS X

Compilazione in C: GCC

- Compiliamo il codice sorgente in hello.c usando il comando gcc

```
gcc hello.c
```

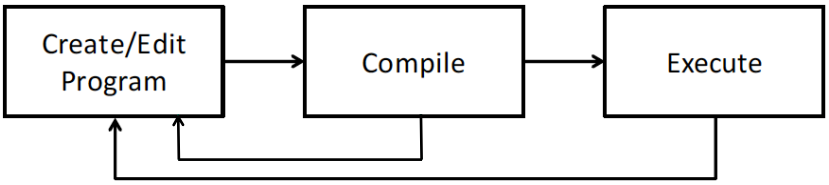
- Il compilatore, se non vi sono errori sintattici, produce il file eseguibile usando il nome di default a.out
- Per eseguire il codice:

```
./a.out
```

- Per specificare il nome del file eseguibile, usiamo l'opzione -o (output)

```
gcc -o hello hello.c
./hello
```

Il processo di programmazione in C



- Il ciclo termina quando il programmatore è soddisfatto del programma, ad es. prestazioni e correttezza

Paradigmi di linguaggi di programmazione

- Funzionale
  - Tratta la computazione come valutazione di funzioni matematiche (ad es. Lisp, Scheme, Haskell, Scala)
  - Supporta funzioni di ordine superiore
- Imperativo
  - Descrive la computazione in termini di dichiarazioni che modificano lo stato del programma (ad es. FORTRAN, BASIC, Pascal, C)
- Logico (dichiarativo)
  - Esprime la logica della computazione senza descrivere il suo flusso di controllo (ad es. Prolog)
- Orientato agli oggetti
  - Utilizza "oggetti" - strutture di dati costituite da campi e metodi insieme alle loro interazioni - per progettare programmi (ad es. C++, Java, C#, Python)

Esempio: Hello World

Codice sorgente: hello.c

```
#include <stdio.h>

int main(void) {
   printf("Hello World\n");
   return 0;
}
```


Esempio: osservazioni

- Il programma è costituito da direttive e istruzioni
  - Le direttive sono ordini rivolti al compilatore
- `#include <stdio.h>`
  - E' una direttiva che chiede al compilatore di includere l'header file stdio.h che contiene una serie di definizioni relative alle operazioni predefinite di I/O
- `int main () {...}`
  - È la definizione della funzione main che racchiude il corpo principale del programma, contenuto tra i due simboli { e }
- Ogni programma C deve contenere una e una sola funzione main
- Diversamente da Python, l'indentazione non ha un significato sintattico
  - Viene usata solo per migliorare la leggibilità del codice

Esempio: osservazioni

- `printf("Hello World\n");`
  - Istruzione che stampa a schermo la stringa racchiusa tra virgolette
  - `printf` è una funzione predefinita, inclusa a seguito della direttiva `#include <stdio.h>`
  - Le istruzioni sono terminate da ;
  - \ è il carattere di escape o controllo (non è visualizzato a schermo)
  - \n (ovvero newline) è una sequenza di escape che posiziona il cursore all'inizio della riga successiva
- `return 0;`
  - È un'istruzione che termina l'esecuzione della funzione main e restituisce il valore 0 (per indicare che il programma termina con successo)

Esempio: warning di compilazione

Codice sorgente: hello_w.c

```
int main(void) {
   printf("Hello World\n");
   return 0;
}
```

- Manca `#include <stdio.h>`
- Cosa accade quando compiliamo?

```
>$ gcc hello_w.c
hello_w.c:5:2: warning: implicitly declaring library function 'printf' with
type
      'int (const char *, ...)' [-Wimplicit-function-declaration]
      printf("Hello World!¥n");
      ^
hello_w.c:5:2: note: include the header <stdio.h> or explicitly provide a
    declaration for 'printf'
1 warning generated.
```

Commenti

- I commenti incominciano con /* e terminano con */
- Ignorati dal compilatore C
- Aumentano la leggibilità del codice (senza esagerare!)
  - Ogni funzione dovrebbe essere preceduta da un commento che ne descrive gli scopi
- Errore tipico: non terminare un commento con */
  - Rimuovendo */ dalla riga 6 di hello_c.c e compilando

```
>$ gcc hello_c.c
hello_c.c:12:12: warning: '/*' within block comment [-Wcomment]
      return 0; /* indica che il programma è terminato con successo */
               ^
hello_c.c:14:1: error: extraneous closing brace ('}')
} /* fine della funzione main */
^
1 warning and 1 error generated.
```


Esempio: aggiungiamo i commenti

Codice sorgente: hello_c.c

```
/* Un primo programma in C */

#include <stdio.h>

/* La funzione main è il punto d'inizio dell'esecuzione
   del programma */
int main(void)
{
   printf("Hello World!\n");
   return 0; /* indica che il programma è terminato con successo */
} /* fine della funzione main */
```

Dichiarazione di variabili

- C richiede la dichiarazione esplicita delle variabili prima del loro utilizzo
  - Si informa il compilatore sulla variabile prima che la variabile sia effettivamente utilizzata
- In C la dichiarazione della variabile definisce il tipo della variabile
- Una variabile è una locazione (o posizione) nella memoria del calcolatore usata per la memorizzazione di un valore
- Ogni variabile è caratterizata da:
  - Nome
  - Tipo
  - Valore
  - Indirizzo

Dichiarazione di variabili

- Per dichiarare una variabile occorre specificare:
  - Tipo della variabile, almeno un carattere di spazio, nome della variabile e infine ;
  - Es.: `int integer1;`
  - Es.: `double x;`
- C è un linguaggio a tipizzazione statica
  - Diversamente da Python, che è a tipizzazione dinamica
- La mancata dichiarazione di una variabile prima di usarla causa un errore in fase di compilazione
- Approfondiremo nella prossima lezione

Operatore di assegnamento

- Istruzione di assegnamento: simile a Python
- Operatore di assegnamento =
  - Es.: `integer1 = 0;`
- Il valore può essere assegnato alla variabile anche in fase di dichiarazione
  - Es.: `int i = 0;`
  - Es.: `double x=3;`
    - ✓ x memorizzerà `3,0` anziché `3` perché x è dichiarata di tipo `double` (numero reale a doppia precisione)

Operatori aritmetici

- Gli operatori aritmetici sono tutti operatori binari
- Addizione: +
- Sottrazione: -
- Moltiplicazione: *
- Divisione: /
  - La divisione tra interi restituisce un risultato intero
  - Es: 8/5 restituisce 1
- Resto: %
  - Es: 8%5 restituisce 3
- Le regole di priorità degli operatori aritmetici sono le stesse adottate in algebra

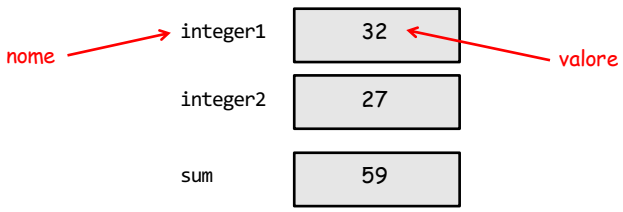
Esempio: addizione

Codice sorgente: sum.c

```
/* Addition program */
#include <stdio.h>

int main(void)
{
   int integer1, integer2, sum;      /* declaration */

   printf("Enter first integer: ");  /* prompt */
   scanf("%d", &integer1);           /* read an integer */
   printf("Enter second integer: "); /* prompt */
   scanf("%d", &integer2);           /* read an integer */
   sum = integer1 + integer2;/* assignment of sum */
   printf("Sum = %d\n", sum);/* print sum */

   return 0;  /* indicate that program ended successfully */
}
```


Esempio: osservazioni

- `scanf("%d", &integer1);  /* read an integer */`
  - Istruzione di lettura: il valore digitato dall'utente è memorizzato nella locazione di memoria alla quale è stato assegnato il nome `integer1`
- Il processo di scrittura in una locazione di memoria è distruttivo
- Le locazioni di memoria dopo l'addizione:



Operatori di uguaglianza e relazionali

- Operatori di uguaglianza
  - Uguale: ==
  - Diverso: !=
- Operatori relazionali
  - Maggiore: >
  - Minore: <
  - Maggiore o uguale: >=
  - Minore o uguale: <=
- Attenzione a non confondere l'operatore di uguaglianza (==) con quello di assegnamento (=)
- Attenzione a non lasciare spazi tra i due simboli che compongono alcuni operatori (es. !=, >=)
  - Errore segnalato dal compilatore

Esempio: operatori algebrici e relazionali

Codice sorgente: operators.c

```
/* Using if statements, relational
   operators, and equality operators */
#include <stdio.h>

main()
{
   int num1, num2;

   printf("Enter two integers, and I will tell you "
          "the relationships they satisfy");
   scanf("%d %d", &num1, &num2);    /* read two integers */

   if (num1 == num2)
      printf("%d is equal to %d\n", num1, num2);

   if (num1 != num2)
      printf("%d is not equal to %d\n", num1, num2);
```

Esempio: operatori algebrici e relazionali (2)

```
    if (num1 < num2)
       printf("%d is less than %d\n", num1, num2);

    if (num1 > num2)
       printf("%d is greater than %d\n", num1, num2);

    if (num1 <= num2)
       printf("%d is less than or equal to %d\n",
              num1, num2);

    if (num1 >= num2)
       printf("%d is greater than or equal to %d\n",
              num1, num2);

    return 0;   /* indicate program ended successfully */
}
```


Esempio: osservazioni

- Istruzione `if` in cui viene valutata una condizione
  - Se la condizione è vera, viene eseguita l'istruzione presente all'interno del corpo dell'istruzione `if`

Esempio: indentazione

- Altra differenza evidente tra C e Python

C fragment

```
disc = b * b - 4 * a * c;
if (disc < 0)
{
    num_sol = 0;
}
else
{
    t0 = -b / a;
    if (disc == 0)
    {
        num_sol = 1;
        sol0 = t0 / 2;
    }
    else
    {
        num_sol = 2;
        t1 = sqrt(disc) / a;
        sol0 = (t0 + t1) / 2;
        sol1 = (t0 - t1) / 2;
    }
}
```

Python equivalent

```
disc = b * b - 4 * a * c
if disc < 0:
    num_sol = 0
else:
    t0 = -b / a
    if disc == 0:
        num_sol = 1
        sol0 = t0 / 2
    else:
        num_sol = 2
        t1 = disc ** 0.5 / a
        sol0 = (t0 + t1) / 2
        sol1 = (t0 - t1) / 2
```