Linguaggi di Programmazione 2022-2023
Introduzione a C e C++ (1)

Marco Antoniotti
Gabriella Pasi
Fabio Sartori

Introduzione al C (ed al C++)

- Il C è uno dei linguaggi fondamentali
  - Linguaggio base per l'implementazione di UNIX, e quindi di Linux, e di Windows
  - Linguaggio di livello relativamente basso ("*a glorified assembly language*")
  - Richiede molta disciplina ed aderenza a varie convenzioni
  - Referenze fondamentali: Kernigham e Ritchie, Harbison e Steele
- Il C++ nasce come estensione ad oggetti del C, essenzialmente ispirata a Simula
  - Evolve negli anni
  - Adotta altri paradigmi (template programming)
  - Referenze fondamentali: Stroustroup, Lippman

Una mappa per il C (e parte del C++)

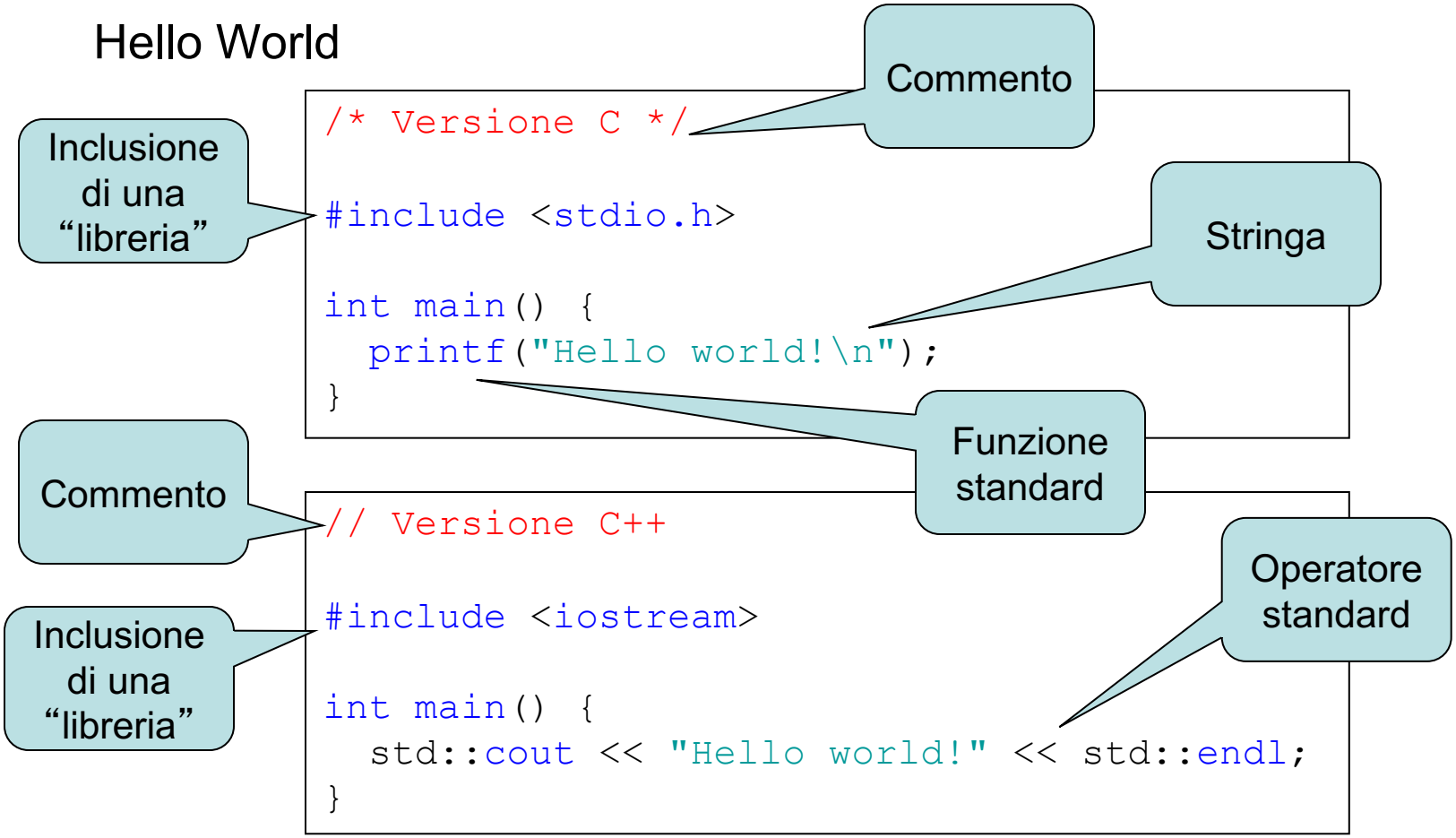
- Linguaggio base
  - Tipi essenziali
    - Corrispondenza con oggetti HW
  - Costrutti sintattici
- Modalità di programmazione
- Strumenti di programmazione
- Librerie standard
- Interazione con il sistema operativo
  - In particolare il sottosistema di I/O

Il più corto programma in C/C++

```
int main() {}
```

Il solito programma

Hello World




```c
/* Versione C */

#include <stdio.h>

int main() {
  printf("Hello world!\n");
}
```

```cpp
// Versione C++

#include <iostream>

int main() {
  std::cout << "Hello world!" << std::endl;
}
```

Compilazione ed esecuzione

- Assunzioni fondamentali
  - Si usano gli strumenti richiamabili dalla linea di comando (*"command line interface"*)
  - L'interprete di comandi è una shell UNIX o il `cmd` di Windows
  - È possibile avere un sottosistema UNIX sotto Windows installando MinGW (http://mingw.org)
- Il compilatore C e/o C++ ha diversi nomi su diverse piattaforme
  - `gcc` (`g++`) è il compilatore disponibile su molti sistemi UNIX (Mac OS X incluso); altri compilatori possono essere `cc` (il nome storico del primo compilatore sulle prime versioni UNIX) o `CC` o `cxx` ecc. ecc.
    - Vari "Integrated Development Environments" (IDEs), ad esempio Eclipse (http://www.eclipse.org, originariamente costruito per Java) di fatto richiamano il compilatore suddetto
  - `cl` è il compilatore Microsoft
    - Visual Studio 2019 è un IDE che, comunque, richiama il compilatore `cl`
    - Installandolo si creano anche dei shortcuts speciali per `cmd` e `Power Shell` che vi permettono di usarlo direttamente

Compilazione ed esecuzione

- Supponiamo di aver salvato il programma C (C++) in un file chiamato `hello.c` (`hello.cc` o `hello.ccx`)
  - Nel seguito, ciò che si batte al "prompt" è in **grassetto**

- Per compilare il programma, si invoca il compilatore

```
prompt$ gcc hello.c
```

- Se non ci sono errori, l'eseguibile sarà un file chiamato (convenzionalmente sulle piattaforme UNIX) `a.out`, che può essere richiamato direttamente

```
prompt$ a.out
Hello world!
```

Compilazione ed esecuzione

- È possibile richiedere un nome specifico per l'eseguibile utilizzando l'opzione `-o`

```
prompt$ g++ -o ciao hello.cc
prompt$ ciao
Hello world
```

- I compilatori C/C++ hanno moltissime opzioni, molte delle quali sono estremamente specializzate
- Diverse opzioni verranno introdotte di caso in caso

Compilazione ed esecuzione

- In seguito vedremo anche I seguenti argomenti
  - Compilazione separata
  - “Linking”
  - Costruzione di librerie
  - Costruzione di sistemi il cui codice sorgente è distribuito in molti files (`make` e simili)

Introduzione al C/C++

- Il C++ è stato definito dal suo creatore, B. Stroustroup "*a better C*"
  - Di fatto è un C esteso
  - Quindi i tipi di dati fondamentali si comportano (essenzialmente) allo stesso modo
    - Possono esistere delle sottili differenze tra C e C++
    - Per capire come queste differenze possono avere delle conseguenze su un programma è necessario far riferimento ai documenti di standardizzazione dei due linguaggi

Introduzione al C: nomi

- In C/C++ i vari elementi del linguaggio (variabili, tipi, classi, funzioni, metodi) sono denotati da nomi (o identificatori)
- In C/C++ i nomi sono stringhe di lettere, numeri, ed il carattere '_'; inoltre devono iniziare con una lettera o con '_'
- Alcuni nomi sono riservati
- Esempi
  - Identificatori validi

```
bho      pi314   un_nome_piuttosto_lungo UnNomeSuEGiu
_____    x
```

  - Identificatori non validi

```
0123     con spazi      $sys    class   if foo~bar .name
```

Introduzione al C: tipi fondamentali

- Il C/C++ ha i seguenti tipi di dato fondamentali
  - `int`, interi da -2^{31} a $2^{31} - 1$ (assumiamo HW con interi di 32 bits)
  - `char`, caratteri; rappresentati come numeri interi da -128 a 127 (ovvero dei bytes da 8 bit)
    - Questi tipi possono essere anche dichiarati `unsigned` per spostare l'intervallo di valori tra 0 e 2^{32} per `int` e 0 e 2^8 per `char`
  - Numeri floating point, `float`
  - Booleani: `bool` (introdotto originariamente nel C++)
  - *puntatori* e *riferimenti* ("*pointers*" e "*references*")
    - La parte più importante, sebbene complicata, dei due linguaggi
- Il C ha due tipi aggregati
  - Arrays (assimilabili a puntatori)
    - `int a[10]` un array di 10 interi
    - `char c[80]` un array di 80 caratteri
    - `float m[2][2]` una matrice floating point 2x2 (di fatto un array di due vettori)
  - Strutture
    - Aggregazioni di "campi" di tipo diverso ("*records*") che vedremo oltre

Introduzione al C: tipi fondamentali

- Il C ha un tipo particolare che denota la mancanza di informazione
  - `void`
- Infine il C ha a disposizione un tipo “enumerazione” di constanti
  - `enum [<nome>] { <c1>, <c2>, …, <cn> }`
- Il C++ ha a disposizione la nozione di classe come base della programmazione object-oriented.

Introduzione al C: variabili

- Un programma C deve manipolare valori che vanno associati a “nomi” ovvero a variabili e/o funzioni
- Tutti questi nomi vengono introdotti in un programma C/C++ per mezzo di dichiarazioni
- Le dichiarazioni - di solito - hanno la forma
  <tipo> <modificatori> <nome> <modificatori> [= <inizializzazione>]
- Esempi:

```
char a = 'a'; /* La variabile a e` di tipo carattere. */
float x;      /* La variabile x e` di tipo 'float'. */
int qd = 42;  /* La variabile qd e` di tipo intero ed e`
                 inizializzata al valore 42. */
int f(float, bool); /* f e` una funzione che prende un
                       float ed un booleano e ritorna un
                       intero. */
```

Introduzione al C: variabili

- Altri Esempi:

```
char* ps;      /* Puntatore ad una locazione di memoria che contiene
                  un carattere. */
float** m;     /* Puntatore ad una locazione di memoria che contiene
                  un puntatore ad una locazione di memoria
                  contenente un 'float'. */
float v[42];   /* Un array di 42 'float'. */
float v2[];    /* Un array di 'float' di dimensione non
                  specificata. */
float m2[42][]; /* Un array di 42 arrays di dimensione non
                   specificata. */
int (*pf)(int*, float*); /* Puntatore ad una funzione che accetta
                            un puntatore ad un intero ed un
                            puntatore ad un 'float' e che ritorna
                            un intero. */
```

- Solo in C++

```
int x = 42;    /* Un intero. */
int& rx = x;   /* Una 'referenza' all'intero x. */
```

Introduzione al C: arrays

- Gli array sono una componente importante del C/C++, ma il loro comportamento è molto sofisticato e, allo stesso tempo, di *"basso livello"*
- Gli array sono da considerarsi come un blocco di memoria fisica
  - La dichiarazione

    ```
    unsigned char s[80];
    ```

    di fatto riserva un blocco di 80 "ottetti" (bytes di 8 bits) nella RAM del calcolatore
  - Si noti come una zona di memoria (diciamo di 512 Mbs) potrebbe essere dichiarata in un programma C come

    ```
    unsigned char total_memory[512 * 1024 * 1024];
    ```

Introduzione al C: arrays

- Gli array non hanno attributi associati, a parte il tipo
  - Non si può estrarre la dimensione dell'array dal contenuto di una variabile
- Un array può essere *inizializzato* con dei valori iniziali letterali

```
int un_due_tre[3] = {1, 2, 3};
```

- In questo caso non è necessario dichiarare la dimensione dell'array

```
float an_array[] = {3.0, 42.0};
```

- Però, bisogna stare attenti

```
float fa[2] = {3.0, 42.0, 0.0};    /* Errore! */
```

Introduzione al C: stringhe

- Le stringhe in C (ed anche in C++) sono degli array terminati con un carattere nullo `'\0'`

```
char stringa[] = "Dylan Dog";
```

Equivale a

```
char stringa[]
     = {'D', 'y', 'l', 'a', 'n', ' ', 'D', 'o', 'g', '\0'};
```

Quindi la lunghezza di questa stringa è 10.

- La convenzione di terminare una stringa con un carattere `'\0'` (che "equivale" anche all'intero `0` - ed in C++ al valore booleano `false`) è fondamentale in C/C++.

Introduzione al C: strutture

- Le strutture in C (e C++) sono aggregazioni di tipi diversi

```
struct polar_complex { float magnitude; float angle; } c
     = { 42.0, 3.14 };

struct persona { char nome[80]; int età; } p
     = { ”Salvo Montalbano”, 42 };
```

- I “campi” di una struttura si estraggono con la notazione punteggiata

```
p.nome
```

ha valore `”Salvo Montalbano”`

- In C++ è possibile usare dei “costruttori” per inizializzare le strutture.

Introduzione al C: puntatori

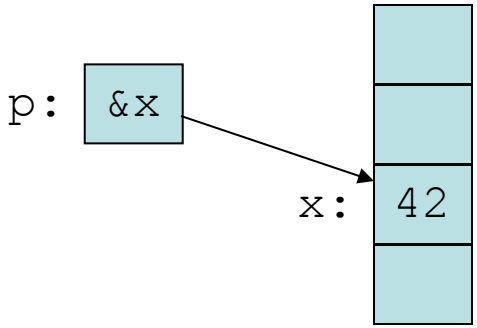
- Il concetto di “puntatore” (o “indirezione” o “riferimento”) è fondamentale per qualunque tipo di programmazione.
- Ogni sistema di programmazione ha - a qualche livello - un concetto equivalente
  - In Java, di fatto, (quasi) tutto è un puntatore
  - Nella programmazione dei DB relazionali, le strutture di chiavi incrociate, non sono altro che una particolare realizzazione di questo concetto
- In C/C++ (così come in Pascal, Ada ed altri linguaggi) puntatori e referenze sono espliciti
- Ciò aggiunge flessibilità al linguaggio, ma con un costo aggiuntivo in complessità

Introduzione al C: puntatori

- Dato un qualunque tipo *T*, il tipo associato *T** è il tipo "puntatore a *T*"
- Ovvero, una variabile di tipo *T** contiene l'indirizzo in memoria di un oggetto di tipo *T*
- **Esempio**

```
int x = 42;
int* p = &x;/* p contiene l'indirizzo di x (ottenuto
             * tramite l'operatore prefisso '&')
             */
```

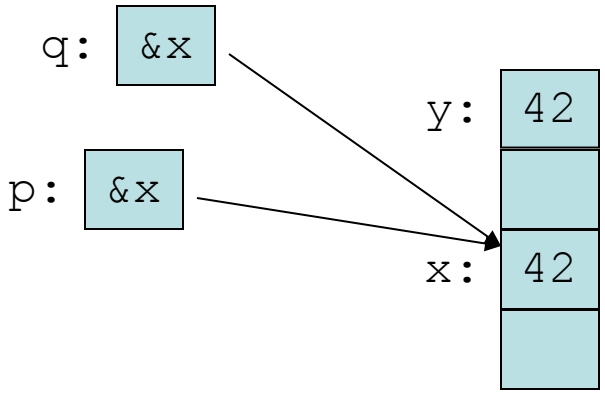
- **Graficamente**



Introduzione al C: puntatori

- L'operazione fondamentale su un puntatore è quella di de-referenziazione ("*dereferencing*")
- **Esempio**

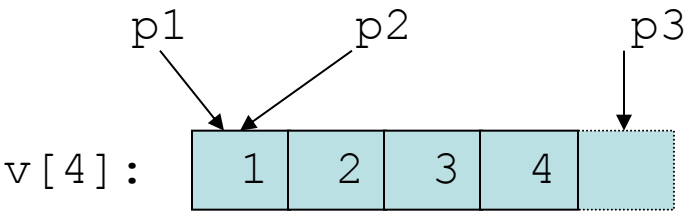
```
int x  = 42;
int* p = &x;
int* q = p; /* assegnamento di un puntatore ad un altro */
int y  = *q;/* de-refenziazione: il valore di y e` 42 */
```



Introduzione al C: puntatori ed arrays

- Un array ed un puntatore sono molto simili in C (ed in C++)
- Il 'nome' di un array può essere usato come un puntatore al suo primo elemento
- **Esempio**

```
int v[] = {1, 2, 3, 4};
int* p1 = v;          /* puntatore al primo elemento
                       *(conversione implicita).
                       */
int* p2 = &v[0];      /* puntatore al primo elemento. */
int* p3 = &v[4];      /* puntatore al primo elemento oltre
                       * l'ultimo.
                       */
```

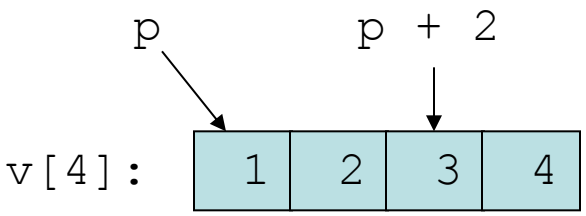


Ovviamente ciò può essere causa di problemi vari, soprattutto di sicurezza

Introduzione al C: puntatori, arrays e aritmetica

- In C/C++ è possibile eseguire operazioni aritmetiche su puntatori
- Ovvero, dato un puntatore `p`, l'espressione `p+2` ha un senso
- Se consideriamo, l'esempio, precedente

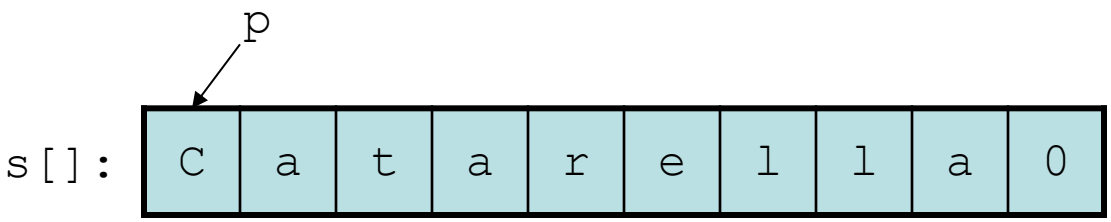
```
int  v[] = {1, 2, 3, 4};
int* p = v;          /* puntatore al primo elemento */
int  x = *(p + 2);   /* dereferenziamo due puntatori oltre
                      * il primo
                      */
int  y = v[2];
bool xy = (x == y); /* xy e` vero */
```



Introduzione al C: puntatori, arrays, aritmetica e stringhe

- Dato che in C/C++ una stringa non è altro che un array terminato da un carattere nullo, e dato che il valore di falsità è collassato sul valore `0`, la funzione di libreria `strlen` che calcola la lunghezza di una stringa può avere il seguente ciclo
- Se consideriamo l'esempio precedente

```
char  s[] = ”Catarella”;
char* p = 0;          /* puntatore a nulla */
int   l = 0;
for (p = s; *p; p++) l++;
/* Alla fine del ciclo, l contiene la lunghezza di s */
```



Introduzione al C: blocchi e operatori condizionali e di iterazione

- In C/C++ vi è il concetto di blocco di operazioni racchiuso tra graffe `{ }`
  - Ogni blocco introduce un ambito ("*scope*")
- Il C/C++ ha i soliti operatori condizionali e di iterazione con la seguente sintassi

```
if (<espressione>)
     <blocco o statement>
[else <blocco o statement>]

while (<espressione>)
     <blocco o statement>

do <blocco o statement>
while (<espressione)

for (<inizio>; <espressione>; <espressione>)
     <blocco o statement>
```

Introduzione al C: blocchi e operatori condizionali e di iterazione

- Altri istruzioni

```
switch (<espressione>) {
case <valore letterale 1>:
     <statement>* [break;]
case <valore letterale 2>:
     <statement>* [break;]
…
case <valore letterale k>:
     <statement>* [break;]
[default:
     <statement>* [break;]]
}
```

Introduzione al C/C++: altri "statements"

- Infine

```
return <espressione>;

goto <label>;
<label> : <statement>
```

- In C++ abbiamo anche

```
try { <statement>* }
catch (<dichiarazione di eccezione>) {
 <statement>*
} …
```

Introduzione al C: espressioni

- In C/C++ le espressioni sono costruite a partire da vari operatori
  - Accesso `. -> [] &` (prefisso)
  - Chiamata di funzione `()`
  - Dimensionamento **`sizeof`**
  - Aritmetici `+ - * / %`
  - Incrementi e decrementi `++ --` (prefissi e postfissi)
  - Shift `<< >>`
  - Logici e booleani `~ ! < <= > >= == != & ^ | && ||`
  - Condizionali *<espressione>* `?` *<espressione>* `:` *<espressione>*
  - Assegnamento `= *= /= %= += -= <<= >>= &= |= ^=`
  - Sequenza `,` (virgola)
- In C++ abbiamo anche altri operatori
  - Informazioni su tipi **`typeid`**
  - Cambiamenti di tipi **`dynamic_cast static_cast reinterpret_cast const_cast`**
  - Eccezioni **`throw`**

Introduzione al C: funzioni

- Un programma C (ed anche C++) è costituito da un insieme di funzioni
- Una di queste funzioni, `main`, ha il ruolo particolare di rappresentare il punto d'inizio di un programma

- Una funzione C (C++) viene definita nel seguente modo

```
<result type> <function name> ( <arg>* ) {
     <declaration>*
     <statement>*
}
```

- Ad esempio

```
int plus(int a, int b) { return a + b; }
```

Introduzione al C/C++: funzioni

Altri esempi:

```
/* La funzione find di QUICK-FIND. */

bool find(int p, int q) {
  return set_id_of[p] == set_id_of[q];
}

/* La funzione fattoriale (con un problema! Anzi tre!). */

int fact(int n) {
  if (n = 0)
    return 1;
  else
    return n * fact(n - 1);
}

/* Oppure (con gli stessi problemi) */

int fact(int n) return n = 0 ? 1 : n * fact(n - 1);
```

Introduzione al C: funzioni e variabili locali

- Abbiamo visto delle funzioni che accettano dei parametri e come i loro tipi vengono dichiarati
  - Non molto diversamente da Java; il che non stupisce, dato che Java è modellato intenzionalmente su C/C++
- Le variabili locali ad un blocco (o ambiente) vengono dichiarate in modo simile

```
int fact_i(int n) {
     int i, acc = 1;

     for (i = 1; i < n; i++) acc *= i;
     return acc;
}
```

- Le variabili locali possono essere inizializzate (ed è bene farlo)

Introduzione al C: funzioni e passaggio di parametri

- Tutti i parametri vengono passati ad una funzione "per valore"
  - Con l'eccezione degli array
- I puntatori (e le referenze in C++) ci permettono di simulare ciò che in altri linguaggi viene chiamato "passaggio di parametri per riferimento"

```
int x = 3;
int y = 42;
void swap(int* a, int* b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
```

- La chiamata

```
swap(&x, &y); /* Notare gli operatori '&' */
```

scambia i valori di `x` ed `y`

Introduzione al C: funzioni e passaggio di parametri

- In C++ la funzione swap può essere scritta usando il tipo "reference"

```
int x = 3;
int y = 42;
void swap(int& a, int& b) {
  int temp = a;
  a = b;
  b = temp;
}
```

- La chiamata

```
swap(x, y);
```

scambia i valori di `x` ed `y`; si noti l'assenza dell'operatore prefisso di indirizzo

Introduzione al C/C++: funzioni e passaggio di parametri

- In C/C++ gli array vengono passati ad una funzione per riferimento
- Ovvero, un parametro di tipo array converte automaticamente un espressione di tipo `T[]` (un array) ad un puntatore al primo elemento.

```
int i[3] = {1, 0, 0};
int v[3] = {1, 2, 3};

int vsmult(int a[], int b[]) {
  int result = 0;

  for (i = 0; i < 3; i++) result += a[i] * b[i];
  return result;
}
```

- La chiamata

```
int s = vsmult(i, v);
```

deposita il valore `1` in `s`, senza copiare in `a` e `b` gli interi arrays `i` e `v`

Introduzione al C: funzioni e passaggio di parametri

- Passare parametri per riferimento (o con un puntatore o, in C++, con una referenza) è molto utile anche per le strutture

```
struct persona {
  char nome[80];
  int eta;
  char assistente_fidato[80];
} p = {”Salvo Montalbano”, 42, ”Catarella”};
```

- Una variabile di tipo `struct persona` ha una dimensione di almeno 164 ottetti (bytes di 8 bits)
- Se vogliamo scrivere una funzione che manipoli il valore della variabile `p`, sarebbe bene usare dei puntatori

```
void stampa_persona(struct persona* x) {
  printf(”<Persona ’%s’ ’%s’ %d>\n”,
         (*x).nome,
         x->assistente_fidato,
         x->eta);
  /* L’operatore ’p->q’ e` un abbreviazione di (*p).q */
}
```

Introduzione al C: funzioni e passaggio di parametri

- La chiamata

```
stampa_persona(&p);
```

produce in output

```
<Persona 'Salvo Montalbano' 'Catarella' 42>
```

NB: i più recenti standard C permettono il passaggio di strutture "per valore".

Riassunto: passaggio di parametri

- In letteratura ed in pratica esistono parecchie modalità di passaggio di parametri a funzioni
- I più importanti sono
  - Passaggio per valore
  - Passaggio per riferimento
- Altre modalità sono
  - Passaggio per "nome"
  - Passaggio per copia-in-out
- In C tutti i parametri sono passati per valore (con l'eccezione degli array)
  - Il passaggio per riferimento è "simulato" usando i puntatori
- Il C++ ammette anche una modalità per "referenza" che, di fatto, lo rende simile ad altri linguaggi

Conclusione

- Brevissima introduzione al C (ed al C++)
- Moltissimi dettagli sono stati sorvolati
- Conoscere questi dettagli è importante
- Leggere e capire uno dei libri di C è molto importante