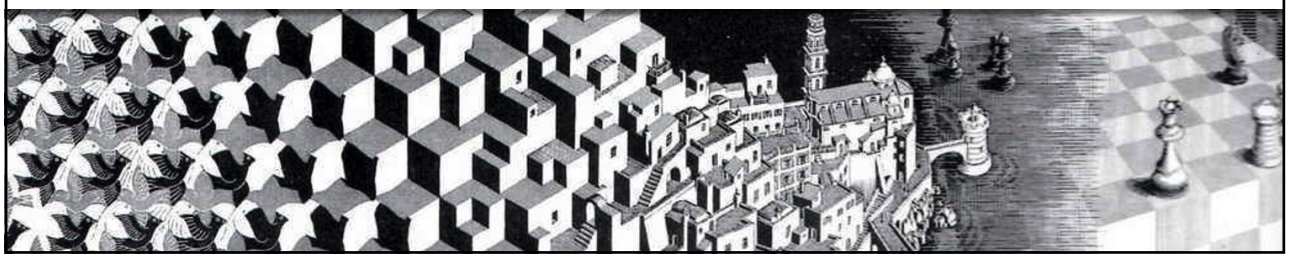
Università degli Studi Roma Tre – Dipartimento di Matematica e Fisica – Corso di Laurea in Matematica

Appunti del corso IN110 Algoritmi e Strutture Dati

6 – Introduzione al linguaggio C

Prof. Marco Liverani

(liverani@mat.uniroma3.it – http://www.mat.uniroma3.it/users/liverani/IN110)



Sommario

- Struttura di un programma
- Direttive del precompilatore, inclusione degli header
- Dichiarazione di variabili
- Operatori aritmetici, logici e di confronto
- Definizione di funzioni
- Passaggio di parametri alle funzioni
- Struttura condizionale
- Strutture iterative
- Compilazione

Struttura di un programma C 1

- Un programma C è costituito da
  - Inclusione di librerie
  - Definizione di costanti e variabili globali
  - Definizione di funzioni
  - La funzione **main**

- Esempio:

```
#include <stdlib.h>
#define PI 3.1415
float media(int a, int b, int c) {
  …
}
int main(void) {
  …
}
```

Direttive per il precompilatore

Definizione di funzioni

Definizione della funzione **main**

Struttura di un programma C 2

- In C le parentesi graffe delimitano blocchi di istruzioni che fanno parte di un medesimo «contesto» (es.: il corpo di una funzione, le istruzioni da eseguire all'interno di un ciclo, …)
- Ogni istruzione in C termina con un punto e virgola «**;**» (esistono però numerose eccezioni)
- Il C è case sensitive: il compilatore è sensibile alla differenza fra lettere maiuscole e minuscole; quasi tutte le parole chiave del linguaggio sono scritte con caratteri minuscoli
- Ogni programma C contiene una funzione **main** da cui ha inizio l'esecuzione del programma stesso
- I commenti (una porzione di testo presente nel programma che il compilatore ignora) sono delimitati dai simboli «**/***» e «***/**»

Direttive per il precompilatore 1

- Il linguaggio C è molto povero: è costituito da poche istruzioni
- È però arricchito da numerose librerie in cui vengono definite funzioni e strutture dati che è possibile utilizzare nei programmi
- Per usare tali funzioni prima di iniziare la compilazione del programma il precompilatore deve caricare le librerie
- Per indicare quali librerie devono essere caricate si utilizza la direttiva **`#include`**:
  - Es.: **`#include <stdio.h>`** carica la libreria *STandarD Input Output*
  - Es.: **`#include <math.h>`** carica la libreria delle funzioni matematiche
- Nei file «**`*.h`**» vengono definiti i «prototipi» delle funzioni (il nome, i parametri ed il valore restituito dalle funzioni)

Direttive per il precompilatore 2

- Mediante la direttiva del precompilatore **`#define`** è possibile definire dei simboli che assumeranno il valore di costanti all'interno del programma
- Il precompilatore (quindi prima che venga avviata la traduzione del programma in linguaggio macchina da parte del compilatore) sostituisce ogni occorrenza dei simboli definiti con i rispettivi valori
- Es.: `#define MAX 100` definisce la "costante" MAX=100

  `#define PI 3.1415` definisce la "costante" PI=3.1415

Tipi di dato

- In C sono a disposizione del programmatore numerosi **tipi di dato elementare** con cui definire variabili e funzioni
- Ad esempio:

Numeri interi	`int`	Interi relativi
	`unsigned`	Interi senza segno
	`long`	Interi lunghi
	`short`	Interi corti
Numeri razionali	`float`	Numeri floating point in singola precisione
	`double`	Numeri in doppia precisione
Caratteri	`char`	Caratteri alfanumerici

- La parola chiave **`void`** indica il tipo *nullo*, ovvero l'insieme vuoto

Dichiarazione delle variabili 1

- **Dichiarando una variabile** si comunica al compilatore il **tipo di dato** che intendiamo associare ad una variabile di memoria e il **nome** con cui nel programma si farà riferimento all'area di memoria in cui è stata «allocata» la variabile stessa
- Questo è fondamentale per consentire alla macchina di **allocare** (riservare) la giusta quantità di memoria e di trattare correttamente la rappresentazione binaria del dato memorizzato (una stessa sequenza di bit può rappresentare informazioni completamente diverse a seconda della *convenzione* che si adotta per interpretarla)
- Una variabile dichiarata in una funzione è **locale** a quella funzione: non può essere utilizzata direttamente fuori dalla funzione in cui è stata definita e viene distrutta non al termine dell'esecuzione della funzione
  - due variabili definite con lo stesso nome in due funzioni distinte, fanno riferimento a diverse aree di memoria
  - non è possibile definire due variabili con lo stesso nome nella stessa funzione
- È possibile (ma è sconsigliato farne largo uso) dichiarare delle variabili **globali**, definite fuori da ogni funzione e quindi visibili in modo diretto da ogni funzione all'interno di uno stesso programma

Dichiarazione delle variabili 2

- La dichiarazione di una variabile avviene riportando il tipo di dato, seguito dall'elenco delle variabili separate da virgole
  - Es.: `int a, b, c;`
- I nomi delle variabili sono case sensitive; sono costituiti da lettere dell'alfabeto, numeri ed il simbolo «_» (*underscore*)
- Devono iniziare con una lettera dell'alfabeto o il simbolo *underscore*
  - Es.: `pippo`, `q7`, `_lim`, `max_elem`
- Per dichiarare una variabile puntatore in grado di indirizzare una variabile di un determinato tipo, bisogna premettere il simbolo «*» prima della variabile
  - Es.: `int *a;` «`a`» è una variabile puntatore ad un intero

Operatori 1

- Operatori aritmetici: "`+`" somma, "`–`" sottrazione, "`*`" prodotto, "`/`" divisione, "`%`" modulo (resto della divisione intera)
- Operatore di assegnazione (in forma estesa): "`=`"
  - Es.: "`a = b*c;`", "`x = -27.4;`"
- Operatori aritmetici di assegnazione (in forma compatta): "`++`", "`--`" (es.: "`x--;`" decrementa di una unità il valore di `x`, è equivalente a "`x = x-1;`", ma è più efficiente), "`+=`", "`-=`", "`*=`", "`/=`" (es.: "`x += 7;`" è equivalente a "`x = x+7;`")

Operatori 2

- Operatori logici di confronto: "==" (es.: "a == b" è vero se a e b hanno lo stesso valore, falso altrimenti), "<" (minore) "<=" (minore o uguale), ">" (maggiore), ">=" (maggiore o uguale), "!=" (diverso)
- Operatori booleani: L'operatore booleano unario "not" è rappresentato dal simbolo "!". L'operatore "and" è rappresentato da "&&" e "or" è rappresentato da "||"
- Le espressioni logiche vengono valutate da sinistra verso destra e la valutazione di una espressione si interrompe non appena è possibile stabilirne con certezza il valore

Funzioni di libreria per l'input/output 1

- Nelle librerie standard del C (rese disponibili grazie all'inclusione degli header mediante la direttiva `#include`) contengono numerose funzioni per le operazioni di I/O
- La funzione «`printf(…)`» consente di eseguire delle stampe in output (tipicamente sul display del terminale)

```
printf("formato", espressione, espressione, …);
```

  stampa sul terminale le espressioni in accordo con la formattazione espressa dalla stringa di formato
- La stringa «`\n`» indica che il cursore deve andare a capo (*new line*); «`%d`» indica che in quella posizione andrà visualizzato il valore di una certa espressione numerica in formato decimale
- Esempi:

```
printf("ciao\n");
printf("A = %d \n", a);
printf("%2d + %2d = %3d \n", a, b, a+b);
```

Funzioni di libreria per l'input/output 2

- La funzione «scanf(…)» consente di acquisire in input delle informazioni dall'esterno, memorizzandole in variabili definite opportunamente nel programma

  ```
  scanf("formato", indirizzo di memoria, indirizzo di memoria, …);
  ```

  legge in input una riga dal terminale e memorizza i valori immessi dall'utente (separati da spazi o da caratteri di fine riga) nelle variabili il cui tipo è descritto nella stringa di formato ed il cui indirizzo di memoria è indicato nell'elenco successivo
- L'operatore «&» applicato ad una variabile, restituisce l'indirizzo di memoria in cui è stata allocata la variabile stessa (es.: «&a » è l'indirizzo di memoria in cui è allocata la variabile «a»)
- La stringa di formato è costituita da sequenze «%...» separate da spazi (es.: «%d» per i numeri in notazione decimale, «%c» per i caratteri, «%s» per le stringhe di caratteri, …)
- Esempi:

```
scanf("%d", &n);
scanf("%d %d %d", &a, &b, &c);
scanf("%d %s", &numero, nome);
```

Definizione di funzioni 1

- Un programma C è composto da numerose piccole funzioni che costituiscono l'implementazione di sottoprogrammi
- Ogni funzione è definita specificando:
  - il nome della funzione
  - il dominio della funzione, ossia i parametri che accetta come argomento
  - il codominio della funzione, ossia l'insieme in cui restituisce dei valori
  - il corpo della funzione che descrive le operazioni effettuate dalla funzione stessa

Definizione di funzioni 2

- Esempio: funzione per il calcolo del massimo fra due numeri *floating point*
- Implementazione in C:

```
float max(float a, float b) {
  float m;
  if (a>b)
    m=a;
  else
    m=b;
  return(m);
}
```

- La funzione `max`, in termini formali può essere definita come max: $\mathbb{Q}\times\mathbb{Q} \rightarrow \mathbb{Q}$
  se con $\mathbb{Q}$ abbiamo indicato l'insieme dei numeri razionali, ossia, impropriamente, il tipo di dato `float` (o `double`)

Definizione di funzioni 3

- Fra i vari tipi di dato `void` indica il tipo nullo: viene utilizzato per specificare che il dominio o il codominio di una funzione sono *vuoti*
- Es.: una funzione priva di argomento che non restituisce alcun valore

```
void saluta(void) {
    printf("Ciao!\n");
    return;
}
```

- Se la funzione non restituisce valori, l'istruzione `return` è priva di argomento e le parentesi tonde possono essere omesse

- Attenzione: stampare in output qualcosa **non** equivale a restituire un valore; leggere in input un dato **non** equivale a ricevere un parametro come argomento

Passaggio di parametri 1

- Il passaggio di parametri ad una funzione avviene sempre solo per valore
- Esempio:

```
int max(int a, int b) {
  if (a>b)
    return(a);
  else
    return(b);
}
```

```
int main(void) {
  int x, int y;
  …
  z = max(x,y);
  …
}
```

- È possibile passare esplicitamente il valore di un puntatore in modo da fornire ad una funzione l'indirizzo di memoria di una variabile definita in un'altra funzione

Passaggio di parametri 2

- L'operatore «&» applicato ad una variabile restituisce l'indirizzo di memoria in cui tale variabile è allocata
- L'operatore «*» applicato ad un puntatore (un indirizzo di memoria) restituisce il *valore* memorizzato in tale locazione
- Se da una funzione f_1 passo l'indirizzo di memoria di una variabile locale ad una funzione f_2, allora f_2 sarà in grado di modificare il valore di tale variabile
- Esempio:

```
int a, *b, c;
b = &a;
c = *b;
```

equivale a

```
int a, c;

c = a;
```

Passaggio di parametri 3

- Esempio: funzione per stampare il massimo fra due numeri interi

```
void stampa_max(int a, int b) {
   if (a>b)
      printf("il massimo e' %d.\n", a);
   else
      printf("il massimo e' %d.\n", b);
   return();
}
…
int main(void) {
   int x, y;
   …
   stampa_max(x, y);
   …
}
```

Non è necessario passare l'indirizzo di **x** e **y**: la funzione **stampa_max** dovrà solo stampare il valore massimo, senza modificarlo

È sufficiente quindi passare a **stampa_max** i valori di **x** e di **y**

Passaggio di parametri 4

- Esempio: funzione per scambiare il valore di due variabili
- Dopo aver definito due variabili in una funzione (es.: **main**) voglio invocare un'altra funzione che scambi i valori delle due variabili locali

```
void scambia(int *a, int *b) {
   int c;
   c = *a;
   *a = *b;
   *b = c;
   return();
}
…
int main(void) {
   int x, y;
   …
   scambia(&x, &y);
   …
}
```

È necessario passare a **scambia** l'indirizzo in cui **x** e **y** sono state allocate

La funzione **scambia** infatti dovrà accedere ad una porzione di memoria allocata da **main** e modificarne il contenuto

Struttura condizionale

- Il C agevola l'uso della programmazione strutturata
- Le tre strutture previste (sequenziale, condizionale ed iterativa) sono implementate in modo semplice mediante delle strutture di controllo
- La struttura condizionale è implementata dall'istruzione

```
if (condizione) {
   …
} else {
   …
}
```

- Esempio:

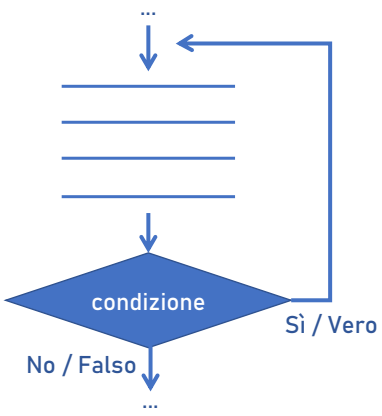
```
if (a > b) {
   ...
}
```

- Se la condizione (costituita da un'espressione booleana) risulta vera allora viene eseguito il primo blocco di istruzioni, altrimenti il blocco che segue la parola chiave **else**
- In ogni caso il programma prosegue poi l'esecuzione con le istruzioni successive al blocco **if … else …**

Struttura iterativa (do … while)

- Esistono in C tre istruzioni differenti per implementare la struttura iterativa
- La prima è l'istruzione **do … while**
- In questo caso la condizione viene verificata alla fine del ciclo: il ciclo viene ripetuto fintanto che la condizione risulta vera; quando diventa falsa il ciclo viene interrotto
- Il blocco di istruzioni viene quindi eseguito sempre almeno una volta, anche se la condizione dovesse essere falsa a priori

```
do {
  …
} while(condizione);
```



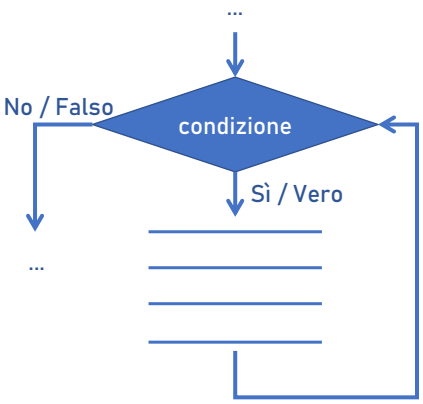
Struttura iterativa (while…)

- La seconda struttura di controllo iterativa è realizzata con l'istruzione **while**:

```
while(condizione) {
  …
}
```

- Le istruzioni del blocco vengono ripetute fintanto che la condizione (espressione booleana) risulta essere vera
- Se la condizione è falsa prima ancora di iniziare il ciclo, non viene eseguita neanche una iterazione
- Esempio:

```
while (i < n && k != 0) {
    ...
}
```



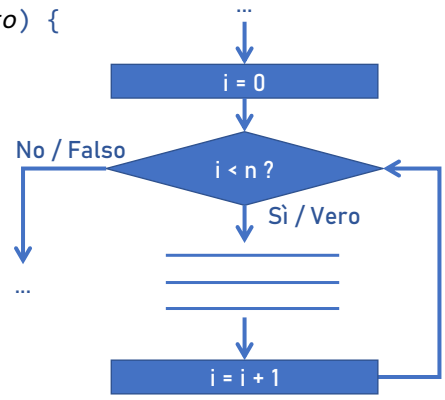
Struttura iterativa (for …)

- La terza struttura di controllo per implementare la struttura iterativa è l'istruzione **for …**
- Questa istruzione è molto comoda (e naturale) quando l'esecuzione del ciclo è controllata da una variabile (un contatore, ad esempio) il cui valore varia da un certo valore iniziale ad uno finale, subendo un incremento (o un decremento) ad ogni iterazione del ciclo

```
for (assegnaz. iniziale; condiz. finale; incremento) {
   …
}
```

- Il ciclo viene iterato fintanto che la condizione risulta essere vera; l'istruzione di incremento viene eseguita al termine di ogni iterazione, prima di verificare la condizione finale
- Esempio:

```
for (i=0; i<n; i++) {
    …
}
```



Compilazione in ambiente UNIX/Linux

- Il compilatore C viene invocato con il comando «`cc`» (C Compiler) o «`gcc`» (GNU C Compiler)
- L'opzione "`-o`" serve per specificare il nome del file eseguibile di output
- L'opzione "`-l`" serve per specificare i nomi di librerie aggiuntive da *linkare* al programma
- Es.: compila il file sorgente «`prova.c`» generando il file eseguibile «`prova`»:

```
cc prova.c -o prova
```

- Es.: compila i due sorgenti «`primo.c`» e «`secondo.c`» generando il file eseguibile «`pippo`» e visualizzando tutti i messaggi di «warning»:

```
cc primo.c secondo.c -Wall -o pippo
```

- Es.: compila «`primo.c`» e «`secondo.c`» collegando anche la libreria matematica (il sorgente contiene la direttiva "`#include <math.h>`") e produce in output il file eseguibile «`pippo`»:

```
cc primo.c secondo.c -o pippo -Wall -lm
```