Università degli Studi di Ferrara

Corso di Laurea in Matematica - A.A. 2021 - 2022

Programmazione
Lezione 5 – Controllo di Flusso

Docente: Michele Ferrari - michele.ferrari@unife.it

Nelle lezioni precedenti

- Il linguaggio C nasce come linguaggio di alto (ma non troppo) livello negli 70 ad opera di Dennis Ritchie
- E’ uno dei linguaggio di programmazione più diffusi ed ha generato derivati massivamente utilizzati nella produzione dei software moderni
- Statico – Non garbage collected – Mutabile – Procedurale – Strutturato
- Codifica di un programma C: editing – compilazione – esecuzione
- Le variabili: porzioni di memoria a cui viene assegnato un nome ed un tipo
- La funzione main: punto di partenza per l’esecuzione del programma
- Output di un programma: la funzione printf

In questa lezione

- Costanti in C
- Operatori
- Operatori Logici
- Funzione scanf()
- Selezione in C
- Iterazione in C

Ripasso: Variabili - Dichiarazione

Il linguaggio C è un linguaggio tipizzato, per questo motivo, prima di utilizzare una variabile è necessario dichiarare che intendiamo utilizzarla attraverso un'operazione detta

dichiarazione della variabile

Con l’operazione di dichiarazione della variabile, definiamo il suo nome ed anche il suo tipo.

La sintassi per dichiarare una variabile è la seguente:

```
<tipo-variabile> <nome-variabile>;
```

Ripasso: Variabili - Tipi di Dato

- Int (e.g. → 5)
- Float (e.g. → 3.4)
- Double (e.g. → 3.5678890)
- Char (e.g. → m)

Nella realtà queste dimensioni sono fortemente dipendenti dalla architettura, e sono definite nel file limits.h in cui sono definiti i valori massimi e minimi per ogni tipo di dato.

Costanti

Una costante è del tutto simile ad una variabile, ovvero è uno spazio di memoria identificato da un tipo e da un nome, come però suggerisce il nome la costante è immutabile: il suo valore non cambia.

Per utilizzare una costante si dichiara una variabile e la si rende costante contestualmente alla dichiarazione, ovvero si aggiunge alla dichiarazione della variabile il modificatore const:

```
const <tipo> <nome> = <valore>;
```

Ad esempio:

```
const float pi = 3.14;
```

Il compilatore segnalerà come illegale qualsiasi tentativo di modificare il valore di una costante, pertanto ogni costante dichiarata con il modificatore const **deve essere inizializzata contestualmente alla dichiarazione.**

Costanti

Nel linguaggio C, esiste un altro modo di definire un valore costante, attraverso la direttiva al precompilatore **`#define`**:

```
#define <stringa> <stringa-di-sostituzione>
```

Non ha tipo, perché questa è una direttiva del precompilatore che provvederà a **sostituire** in tutto il nostro codice ogni occorrenza di stringa con il valore di sostituzione.

Alcuni esempi comuni:

```
#define TRUE 1
#define FALSE 0
#define pi 0.314
```

Costanti

Quando vengono utilizzate?

- Le costanti sono un ottimo strumento per parametrizzare il codice: si può modificare il valore di ogni ricorrenza di una costante nel codice **modificando una sola riga**

- Le costanti posso anche aiutare a migliorare la leggibilità

Gli operatori

In C (e in molti linguaggi) vengono comunemente utilizzati simboli per richiamare alcune operazioni fondamentali, questi vengono detti operatori.

Operatori

Simbolo	Operazione
+	addizione
-	sottrazione
*	moltiplicazione
/	divisione intera
%	modulo

- esempio operatori

Operatore di incremento

Ricordiamo che:

Gli accumulatori, nel linguaggio della programmazione sono variabili nelle quali il nuovo valore non sostituisce il valore precedente, ma si somma (accumula) a quello già presente.

Se all'accumulatore viene "passata" la seguente sequenza:

1, 2, 3, 4

Allora l'accumulatore assumerà i seguenti valori***** :

1, 3, 6, 10

Se il valore di incremento dell'accumulatore resta costante di una unità, allora otteniamo un **Contatore**

***** Supponendo che il contatore sia inizializzato a 0

Operatore di incremento

In ambito di programmazione si fa massivo utilizzo di accumulatori e contatori, per agevolarne l'implementazione in C è presente un set di operatori appositi, il più comune dei quali è

++

Operatore di incremento

Si tratta di un operatore speciale per incrementare una variabile di una unità.

Scrivere:

```
count++;
```

equivale a scrivere:

```
count = count +1;
```

L'operatore “++” è pertanto l'operatore di auto incremento.

Operatore di decremento

Analogamente è possibile utilizzare l'operatore speciale "--"

Scrivere

```
count--;
```

Equivale a scrivere

```
count = count-1;
```

L'operatore "--" è pertanto l'operatore di auto decremento

Pre e Post incremento

L'operatore di autoincremento può essere posizionato prima (pre) o dopo (post) la variabile:

```
count++;
++count;
```

La posizione dell'operatore rispetto alla variabile ha la sua importanza anche se in generale le 2 forme sono equivalenti: bisogna fare attenzione a come lo si usa **nelle istruzioni di assegnamento**

Post Incremento:

```
int count, result;
count = 0;
result = count++;
```

In questo caso al termine di esecuzione, count assumerà il valore 1 mentre result avrà il valore 0.
In altri termini l'operazione di incremento avviene DOPO l'operazione di assegnazione

Pre Incremento:

```
int count, result;
count = 0;
result = ++count;
```

In questo caso al termine di esecuzione, sia count che result avranno il valore 1.

In altri termini l'operazione di incremento avviene PRIMA dell'operazione di assegnazione

Operatore di incremento con valore arbitrario

Se volessimo incrementare il contatore di un valore arbitrario possiamo utilizzare l'operatore +=

Scrivere:

```
count += 2;
```

equivale a scrivere:

```
count = count +2;
```

L'operatore += è pertanto l'operatore di assegnazione in quanto ha il compito di sommare il valore alla sua destra alla variabile alla sua sinistra:

km += 37;

k1 += k2;

a += (b/2);

Operatori di assegnazione

Analoghi a quello appena visto esistono diversi operatori di assegnazione, in modo da poter utilizzare sostanzialmente tutti gli operatori aritmetici:

+=

-=

*=

/=

Funzione scanf()

Abbiamo visto come la funzione printf() venga utilizzata per visualizzare il valore contenuto nelle variabili a schermo (standard output)

Analogamente è possibile acquisire da standard input (l'utente e la sua tastiera) il valore di una variabile, per fare questo si utilizza la funzione scanf()

Funzione scanf()

La funzione scanf() interrompe il flusso del programma e attende che l'utente inserisca un valore da tastiera.

Utilizzo:

```
int base;
printf("Indica il valore per la base: ");
scanf("%d", &base);
```

Il simbolo **&** prima del nome della variabile è **l'operatore di referenziazione** e indica che il valore letto va memorizzato nell'area di memoria identificata dal nome **base**

- In questo esempio `scanf()` legge un valore inserito da tastiera, lo converte in intero poiché abbiamo indicato `%d` e lo memorizza nello spazio di memoria `&base`, ovvero lo spazio di memoria che viene richiamato quando utilizzeremo il nome della variabile (base) nelle successive operazioni

- esempio scanf

Operatore di confronto

L’operatore == è l’operatore di **confronto**

- Restituisce il valore logico TRUE (1) se il valore dell’espressione che sta alla destra dell’operatore è uguale al valore dell’espressione che sta alla sua sinistra
- Restituisce il valore logico FALSE (0) se il valore dell’espressione che sta alla destra dell’operatore è diverso dal valore dell’espressione che sta alla sua sinistra

```
espressione1 == espressione2
```

```
x==y
```

```
(5+3)==(6+2)
```

Operatori Logici

Simbolo	Sintassi	Significato
!	!a	NOT logico
>	a>b	maggiore
<	a<b	minore
>=	a>=b	maggiore/uguale
<=	a<=b	minore/uguale
==	a==b	uguale
!=	a!=b	diverso

Prodotto logico (AND)

a	b	a•b
0	0	0
0	1	0
1	0	0
1	1	1

Somma logica (OR)

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

Selezione in C

Ricordiamo:

Secondo i dettami della programmazione strutturata un buon algoritmo viene espresso in forma di

sequenza – **selezione** – iterazione

Selezione: permette di scegliere fra due alternative la sequenza di esecuzione

Selezione in C: costrutto if()

La selezione in C viene implementata dal costrutto if()

Sintassi:

```
if (condizione) {
  // istruzioni
}
// istruzioni
```

La condizione deve essere una **espressione logica** (restituire vero o falso)

e.g.

`a == b`

`a < b`

`b > 5` ecc..

Selezione in C: costrutto if()

E’ possibile utilizzare gli operatori logici per mettere in relazione più condizioni

Operatore **AND** (&&)

Sintassi

```
if(condizione1 && condizione2){
  // istruzioni
}
```

Operatore **OR** (||)

Sintassi

```
if(condizione1 || condizione2){
  // istruzioni
}
```

Selezione in C: costrutto if()

Nel caso volessimo distinguere due casi **mutuamente esclusivi** possiamo integrare il costrutto if() con il ramo **else**

Sintassi:

```
if (condizione) {
  // istruzioni
} else {
  // istruzioni
}
```

In questo caso i due blocchi di istruzioni vengono eseguiti in maniera alternativa l'uno all'altro dipendentemente dalla condizione

Selezione multipla - else if

Se volessimo implementare una scelta multipla possiamo sfruttare la possibilità di concatenare diversi costrutti “else if”

Sintassi

```
if (condizione1) {

  // istruzioni

} else if (condizione2) {

  // istruzioni

} [...]
```

Iterazione in C

Ricordiamo:

Secondo i dettami della programmazione strutturata un buon algoritmo viene espresso in forma di

sequenza – selezione – **iterazione**

Iterazione: permette di ripetere più volte la stessa sequenza fino al verificarsi di una condizione

Iterazione in C: i cicli

Sono strutture fondamentali utilizzate **per ripetere istruzioni su insiemi di dati diversi**

I linguaggi di programmazione mettono a disposizione vari tipi di cicli per adattarsi alle varie situazioni, in C esistono:

- ciclo “**`while-do`**”
- ciclo “**`do-while`**”
- ciclo “**`for`**”

Il Ciclo While - do

Ciclo iterativo con **controllo in testa**

```
while (condizione) {
  // istruzione
}
```

1. Viene valutata la condizione
2. Se la condizione risulta essere vera, vengono eseguite le istruzioni contenute nel ciclo
3. Si ritorna al punto 1

L'istruzione viene eseguita se e finché la condizione risulta essere vera

Il Ciclo While - do

Esempio:

```
int i=0;
while (i<10) {
   printf("%d", i);
   i++;
}
```

Per eseguire più istruzioni all'interno di un ciclo o di una selezione è necessario racchiuderle all'interno di un blocco {}

Nota: Se l'istruzione è unica, in C possiamo omettere le parentesi

- Esempio while

Il Ciclo do - While

Ciclo iterativo con **controllo in coda**

```
do {
  //istruzioni
} while (condizione);
```

1. Vengono eseguite le istruzioni
2. Verifica della condizione
3. SE vera si riprende dal punto 1, SE falsa si esce dal ciclo

Anche in questo caso l'istruzione viene eseguita finché la condizione risulta essere vera.

A differenza del ciclo while-do siamo però sicuri che **le istruzioni interne al ciclo vengano eseguite almeno una volta**

Il Ciclo For

Il ciclo For viene generalmente utilizzato quando si desidera realizzare un ciclo a contatore, ovvero quando necessitiamo che un insieme di istruzioni venga ripetuto un determinato numero di volte

Sintassi

```
for (inizializzazione; condizione; assegnamento){
  // istruzioni
}
```

1. Viene eseguita l'inizializzazione
2. Viene valutata la condizione
3. Se la condizione risulta essere vera vengono eseguite le istruzioni all'interno del ciclo
4. Dopo l'esecuzione delle istruzioni viene eseguito l'assegnamento (incremento, nella maggior parte dei casi)
5. Si procede dal punto 2

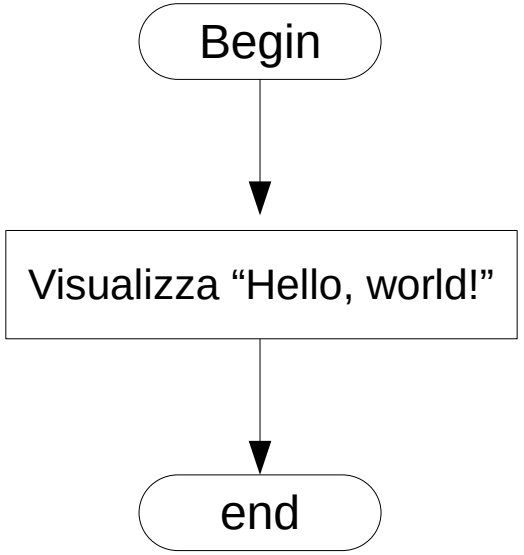
Il Ciclo For

Esempio:

```
int i;
for (i=0; i<10; i++) {
   printf(“%d”, i);
}
```

- esempio for

Hello, world!



Grazie per l’attenzione

Riferimenti

Il corso di programmazione per il primo anno della Laurea Triennale in Matematica nasce con l'intento di unire ai principi di programmazione una conoscenza basilare di uno degli strumenti software più diffusi nell'ambito matematico: Matlab.

La prima parte del corso pertanto è una rielaborazione del programma di Programmazione per la Laurea Triennale in Informatica 15/16 (in particolare) e 16/17.

Parte del materiale originale da cui ho ricavato il percorso didattico, alcuni approfondimenti ed integrazioni possono essere trovati in:

- Lezioni di Programmazione 15/16 CdS Informatica - Giacomo Piva
- Lezioni di Programmazione 16/17 CdS Informatica - Marco Alberti