Lezione 5 e 6

- Concetto di blocco
- Controllo del flusso di un programma
- Costrutti per la scelta if e switch
- Costrutti while e for
- Operatori in C

Fabio Scotti (2004-2009)

Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)

Laboratorio di programmazione

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 5 e 6

Controllo del flusso del programma

Obiettivi :

- Capire come controllare il flusso di un programma mediante la sequenza, la scelta e l'iterazione in C
- Scrivere i primi semplici programmi in C

Concetto di blocco in C

- In C i blocchi sono realizzati con delle parentesi graffe dentro le quali vi è una sequenza di istruzioni
- E' possibile dichiarare delle variabili locali all'interno di in blocco di istruzioni C
- I blocchi possono essere annidati
- Le dichiarazioni delle variabili locali sono nella parte iniziale del blocco

3

Esempio di blocco

```
#include <stdlib.h>
#include <stdio.h>

int main()
{
  int a, b;
  ....
  {
     int temp;   /*  parte dichiartiva del blocco */
     temp = a;
     a = b;
     b = temp;
  }   /*ATTENZIONE da qui in poi la variabile temp cessa di esistere */

}
```

Variabili locali al main (visibili anche nel blocco):

- a, b

Variabile locale al blocco (non visibile fuori):

- temp

4

Regole di scopo e di visibilità

- **Le variabili locali:**
  - sono visibili dalla dichiarazione fino alla fine del blocco cui appartengono
  - rendono non accessibili le variabili omonime globali, cioè dichiarate in blocchi più esterni.

  (dettagli nella lezione sulle funzioni)

- **Ciò che è contenuto fra le graffe del main risponde alla definizione di blocco?**
- **Sì !**
- **Durante il corso non useremo i blocchi contenenti delle dichiarazioni se non per il main**

5

Il costrutto `if` con una istruzione

- **`if` con un'unica istruzione**

```
if (espressione)
  istruzione1
```

- **se `espressione`**
  - è vera allora esegui **`istruzione1`**
  - altrimenti saltala

- **Esempio**

```
if (x==3)
  k=4;
```

6

Il costrutto `if` con un blocco

```
if (espressione)
{
   istruzioni
}
```

- **se** `espressione`
  - è vera allora esegui le **`istruzioni`** nel blocco
  - altrimenti saltale
- **Esempio**

```
if (x==3)
{
    k = 4;
    c = k;
    temp = 3;
}
```

7

Il costrutto `if` con il ramo `else` (1)

```
if (espressione)
    istruzione1
else
    istruzione2
```

- **se** `espressione`
  - è vera allora esegui **`istruzione1`**
  - altrimenti esegui **`istruzione2`**
- **anche in questo caso al posto di una istruzione (k=4;) è possibile inserire un blocco di codice ({k=4; c=k; …}).**

8

Il costrutto `if` con il ramo `else` (2)

- **Esempi:**

```
if (x==3)
   k = 4;
else
   k=0;
```

```
if (x==7)
{
      k = 4;
      temp = 3;
}
else
{
      k = 5;
      c = v;
}
```

9

Il costrutto `if` con 3 o più istruzioni

```
if (espr1)
      istruzione1
else if (espr2)
      istruzione2
else if (espr3)
      istruzione3
else if (espr4)
      istruzione4
```

- **Le istruzioni terminano sempre con un ;**

10

Il costrutto `switch`

- **Il costrutto `switch` è solitamente impiegato quando si devono controllare molti casi.**
- **Il codice infatti risulta essere più ordinato.**
- **E' composto da**
  - delle etichette (**`case`**)
  - un caso generale (**`default`**).
- **Si noti che le etichette del case devono essere delle costanti.**

11

Esempio di costrutto `switch`

```
c = ....;  // sia c il carattere da esaminare
switch (c)
{
     case '0':
          n_cifre = n_cifre + 1;
          break;
     case '1':
          n_cifre = n_cifre + 1;
          break;
     .....
     default:
          n_altri = n_altri + 1;
}
```

12

Comportamento di `switch`

1. **valuta l'espressione di `switch`**
2. **entra nel blocco e**
   1. salta all'etichetta **`case`** a cui è associata una costante uguale al valore calcolato, se è presente, ed esegue le istruzioni corrispondenti
   2. altrimenti salta all'etichetta **`default`**, se presente, ed esegue le istruzioni corrispondenti, altrimenti termina l'esecuzione dello **`switch`.**
3. **quando raggiunge l'istruzione `break` o la fine dell'istruzione `switch` termina l'esecuzione di `switch`.**

13

Il costrutto `while` (1)

- **È il costrutto più generale**
  - da cui si possono ottenere tutti gli altri costrutti per l'iterazione
- **È quello che viene consigliato per il nostro corso.**
- **Il codice infatti risulta essere più ordinato.**
- **Strutture del costrutto `while`**

```
while (espressione)
    istruzione1
```

oppure

```
while (espressione)
{
    istruzioni
}
```

14

Il costrutto `while` (2)

```
while (espressione)
    {
       istruzioni
    }
```

1. **L'`espressione` viene valutata**
   - se ha valore diverso da 0 (*vero*) viene eseguita l'`istruzione` (o il blocco di `istruzioni`)
2. **Una volta che l'esecuzione è terminata, l'`espressione` viene valutata nuovamente**
   - se è nuovamente vera, si ripete l'istruzione.
3. **Ciò si ripete fino a quando l'espressione ha valore 0 (*falso*), nel qual caso il controllo si trasferisce all'istruzione successiva al `while`.**

15

Il costrutto `while` (3)

```
while (espressione)
    {
       istruzioni
    }
```

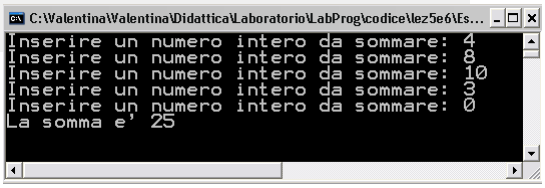
- **Un ciclo `while` può essere eseguito**
  - 0 volte (quando l'espr. è falsa già la prima volta)
  - più volte
- **Si tratta di un ciclo a condizione iniziale: prima di eseguire il ciclo si valuta la condizione**
- **Le strutture iterative possono entrare in ciclo infinito**
  - questo accade se per un errore di programmazione o per qualche condizione esterna l'espressione del `while` rimane sempre vera

16

Esempio di costrutto `while`

- **Problema: sommare i numeri forniti dall'utente tramite tastiera finché non viene immesso uno zero.**

```
#include <stdio.h>
int main()
{
   int i, somma;
   somma = 0;
   i = 1;  // valore iniziale (serve solo per entrare nel while)
   while (i != 0)
   {
     printf("Inserire un numero intero da sommare: ");
     scanf ("%d" , &i);
     somma = somma + i;
   }
   printf("La somma e' %d", somma);
   fflush(stdin);
   getchar();
}
```



17

Il costrutto `for` (1)

- **Il costrutto `for` è equivalente al `while`, ma è da impiegarsi**
  - solo se il numero di iterazioni è noto a priori.
- **Per scandire un struttura dati che ha sempre N elementi useremo un ciclo `for`.**
- **Se invece non sappiamo a priori il numero di iterazioni del ciclo allora impiegheremo un costrutto `while`.**
- **Un programmatore esperto non confonde mai queste due situazioni e usa sempre il costrutto più appropriato.**

18

Il costrutto `for` (2)

```
for ( expr1 ; expr2 ; expr3 )
        istruzione
```

- **Oppure `for` con blocco di codice:**

```
for ( expr1 ; expr2 ; expr3 )
   {
      istruzione1
      istruzione2
      ....
      istruzionen
   }
```

- **Dove:**
  - **`expr1`** è l'espressione iniziale,
  - **`expr2`** è l'espressione Booleana *(del ciclo)*
  - **`expr3`** è l'espressione incremento.

19

Il costrutto `for` (3)

1. **L'espressione iniziale:**
   - permette di inizializzare le variabili di ciclo
   - viene eseguita una volta sola, prima di qualsiasi altra operazione.
   - Esempio expr1: contatore = 0.
2. **L'espressione Booleana (o del ciclo):**
   - viene valutata subito dopo
   - se questa ha valore diverso da 0 viene eseguita l'istruzione che costituisce il corpo del ciclo (o il blocco di istruzioni fra parentesi graffe).
   - Esempio expr2: contatore < 9.

20

Il costrutto `for` (4)

3. **L'espressione-incremento**
   - viene valutata al termine dell'esecuzione del corpo del ciclo
   - serve, di solito, per poter aggiornare i valori delle variabili di ciclo
   - Esempio expr3: contatore = contatore + 1.
4. **Si valuta nuovamente l'espressione del ciclo finché questa non risulta falsa (ovvero uguale a 0)**

- **Esempio:**

```
for ( i=1 ; i<10 ; i++ )
 printf("Indice = %d", i);
```


Scrittura di un ciclo `for` con un ciclo `WHILE`

```
for ( expr1 ; expr2 ; expr3 )
   {
       istruzione1
       ....
       istruzionen
   }
```

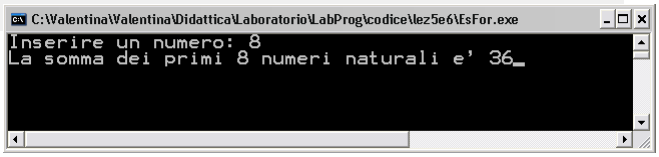
- **è del tutto equivalente a**

```
expr1;
while (expr2)
{
    istruzione1
    ....
    istruzionen
    expr3;
}
```

Esempio di costrutto `for`

- **Problema: chiede all'utente un numero intero N e sommare i primi N numeri naturali.**

```
#include <stdio.h>
int main()
{
   int i, n, somma;
   somma = 0;
   printf("Inserire un numero: ");
   scanf ("%d" , &n);
   for  (i=0; i<= n; i++)
   {
      somma = somma + i;
   }
   printf("La somma dei primi %d numeri naturali e' %d", n, somma);
   fflush(stdin);
   getchar();
}
```



23

Forme compatte

- **Il linguaggio C consente ai programmatori di usare vari stili di scrittura**
- **Le forme compatte sono delle scritture alternative dei normali istruzioni espresse con un minor numero di caratteri:**

```
var op= espressione //forma compatta di:
var = var op espressione
```

- **Esempi:**

`x *= y`	corrisponde a	`x = x * y`
`y -= z+1`	corrisponde a	`y = y - (z + 1)`

- **Le forme compatte rendono il codice più elegante, ma non sempre di facile lettura**
- **Si sconsiglia l'uso delle forme compatte**

24

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 5 e 6

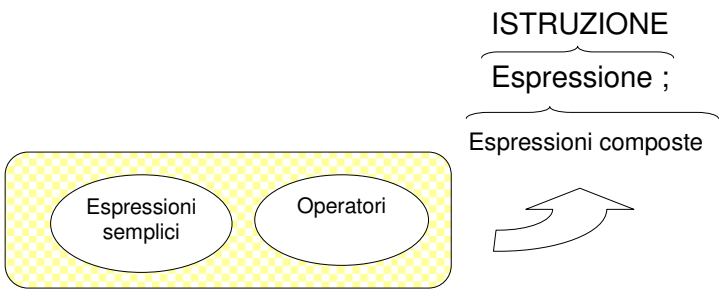
Operatori in C

Obiettivi :

- Conoscere ed iniziare ad utilizzate gli operatori principali del linguaggio C

Operatori in C

- **Gli operatori si dividono in 4 categorie:**
  - operatori di assegnamento;
  - operatori matematici;
  - operatori relazionali;
  - operatori logici.



26

Operatori di assegnamento

- **E' il carattere =**
- **Scrivere `x = y;`**
  - non vuol dire realizzare un confronto
  - ma assegnare il valore di y alla variabile x

Operatori matematici (1)

- **Consentono di effettuare le operazioni aritmetiche**

Operatore	Tipo	Azione	Esempi
`++`	Unario	Incrementa di un'unità	`x++; ++x;`
`--`	Unario	Decrementa di un'unità	`y--; --y;`
`+`	Binario	Somma di due operandi	`x + y;`
`-`	Binario	Sottrazione di due operandi	`y - x;`
`*`	Binario	Moltiplicazione di due operandi	`x * y;`
`/`	Binario	Divisione	`y / x ;`
`%`	Binario	Resto della divisione	`y % x;`

Operatori matematici (2)

- **Precedenza fra gli operatori:**
  1. Incrementi e decrementi unari (++, --);
  2. Moltiplicazioni, divisioni e resti (*, /, %);
  3. Somme e sottrazioni (+ , -);
- **Esempio:**
  - `x = 5 + 4 * 3` assegna 17 (5+12) alla variabile `x`
- **Nel caso di operatori di ugual livello, il compilatore procede da sinistra verso destra:**
  - esempio: `x = 5 * 6 /3` assegna 10 a `x`.
- **Per alterare l'ordine di esecuzione, è necessario introdurre le parentesi**
- **Buona programmazione: usare sempre le parentesi per esplicitare le precedenze**

29

Operatori relazionali

- **Sono utilizzati per confrontare espressioni**
- **Le espressioni relazionali possono essere false (0) o vere (1)**

Operatore	Descrizione	Esempi
==	Confronto di uguaglianza	`x == y`
>	Maggiore di	`x > y`
<	Minore di	`x < y`
>=	Maggiore o uguale di	`x >= y`
<=	Minore o uguale di	`y <= x`
!=	Diverso (x è diverso da y?)	`x != y`

- **L'uso tipico è nelle istruzioni condizionali come if e while, ma si possono usare anche:**
  - `x = (4==4)+(2>3)` equivalente a: `x = 1 + 0`.

30

Operatori logici (1)

- **Sono utilizzati per manipolare i valori Booleani, vero o falso**

Operatore	Descrizione	Esempi
`!`	Not logico	`!x`
`&&`	And logico	`x && y`
`||`	Or logico	`x || y`

- **Nota bene:**
  - OR è || e non |
  - AND è && e non &

31

Operatori logici (2)

- **Operatore di negazione: `!espr`**
  - ritorna 0 se **`espr`** ritorna un valore diverso da 0
  - ritorna 1 se **`espr`** ritorna il valore uguale a 0
- **Operatore di AND: `espr1 && espr2`**
  - Valuta **`espr1`**:
    - se **`espr1`** e' falsa, ritorna 0 senza valutare **`espr2`**
    - se **`espr1`** e' vera, valuta **`espr2`**: Se **`espr2`** e' falsa ritorna 0 altrimenti ritorna 1. Ovvero ritorna **`espr2`**
- **Operatore di OR: `espr1 || espr2`**
  - Valuta **`espr1`**:
    - se **`espr1`** e' vera, ritorna 1 senza valutare **`espr2`**
    - se **`espr1`** e' falsa, valuta **`espr2`**: Se **`espr2`** e' vera ritorna 1 altrimenti ritorna 0. Ovvero ritorna **`espr2`**

32

Operatori bit a bit

- **Operatori che lavorano sui singoli bit delle celle di memoria:**
  - `~` Operatore complemento ad 1
  - `&` Operatore and bit a bit
  - `|` Operatore or (inclusivo) bit a bit
  - `^` Operatore or esclusivo (ex-or) bit a bit
  - `<<` Operatore shift a sinistra
  - `>>` Operatore shift a destra
- **Per semplicità di trattazione, consideriamo gli interi come costituiti da 8 bit, anziché 16 o 32 bit come avviene in realtà sugli elaboratori**

33

Operatore ~

- **`~espressione`**
- **ritorna il complemento ad 1 del valore tornato da espressione.**
- **Il complemento ad 1 consiste nel cambio di ciascun bit con il suo complemento:**
  - ogni bit posto ad 1 viene cambiato a 0
  - ogni bit posto a 0 viene cambiato ad 1
- **Esempio:**

```
int a = 10; // rappresentazione binaria (8 bit): 00001010
int c;
c = ~a; // rappresentazione binaria di c: 11110101
        // in rappresentazione decimale: 235
```

34

Operatore &

- `espr1 & espr2`
- **ritorna il valore dell'AND effettuato bit a bit sui valori ritornati dalle 2 espressioni**
- **Esempio:**

```
int a=10; //rappresentazione binaria (8 bit): 00001010
int b=12; //rappresentazione binaria (8 bit): 00001100
int c;
c = a&b;            /*  00001010    a
                     *  00001100    b
                     *  --------
                     *  00001000    c = a&b */
```

la variabile c contiene il valore 8

35

Operatore |

- `espr1 | espr2`
- **ritorna il valore dell'OR effettuato bit a bit sui valori ritornati dalle 2 espressioni**
- **Esempio:**

```
int a=10; //rappresentazione binaria (8 bit): 00001010
int b=12; //rappresentazione binaria (8 bit): 00001100
int c;
c = a|b;            /*  00001010    a
                     *  00001100    b
                     *  --------
                     *  00001110    c = a|b */
```

la variabile c contiene il valore 14

36

Operatore ^

- **espr1 ^ espr2**
- **ritorna il valore dell'EX-OR effettuato bit a bit sui valori ritornati dalle 2 espressioni**
- **Esempio:**

```
int a=10; //rappresentazione binaria (8 bit): 00001010
int b=12; //rappresentazione binaria (8 bit): 00001100
int c;
c = a^b;            /*  00001010    a
                     *  00001100    b
                     *  --------
                     *  00000110    c = a^b */
```


la variabile c contiene il valore 6

Operatore <<

- **espr1 << espr2**
- **ritorna il valore della traslazione a sinistra di espr2 sui bit del valore ritornato da espr1**
- **I nuovi bit che entrano a destra sono posti a 0**
- **Esempio:**

```
int a=10; //rappresentazione binaria (8 bit): 00001010
int b=2;
int c;
c = a<<b;          // shift a sinistra di 2 bit
                   //  c = 00101000 ovvero c = 40
```

- **Equivale a moltiplicare espr1 per 2^{espr2}**

```
//  c = a * 2^b = 10 * 2^2 = 40
```

Operatore >>

- **espr1 >> espr2**
- **ritorna il valore della traslazione a destra di espr2 sui bit del valore ritornato da espr1**
- **I nuovi bit che entrano a sinistra possono dipendere dall'architettura dell'elaboratore e/o dalla implementazione del compilatore.**
  - Non e' garantito che siano sempre posti a 0
  - Per evitare ciò è bene assicurarsi che il valore ritornato da **espr1** sia di tipo **unsigned**
- **Esempio:**

```
unsigned int a=10; //rappresentazione binaria: 00001010
int b=2;
int c;
c = a>>b;     // shift a destra di 2 bit: 00000010 = 2
```

39