



Informatica Generale

Linguaggio C



Il controllo di flusso

Istruzioni Condizionali

- If
- else
- switch

Cicli

- while do
- do while
- for
- go to





Il controllo di flusso



Il controllo di flusso

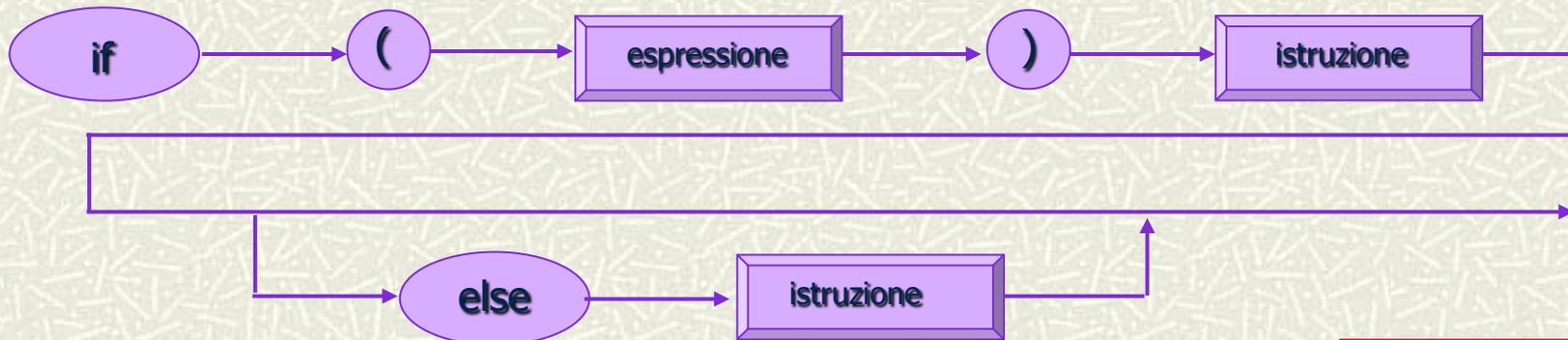
- # Le istruzioni di controllo di flusso si dividono in...
 - ◆ **Selezioni condizionali**
 - ◆ **Cicli**

- # Le **selezioni condizionali** permettono di decidere l'attivazione di parti di codice, in base al valore di un'espressione

- # I **cicli**, o **iterazioni**, consentono di svolgere più volte un insieme di operazioni fino al raggiungimento di una condizione particolare

La selezione condizionale - 1

- ‡ La **selezione condizionale** è la principale modalità di controllo dei linguaggi di programmazione:
 - Permette di prendere decisioni a run-time, attivando o meno una sequenza di istruzioni in dipendenza del valore di un'espressione
 - Dato che il valore di un'espressione può cambiare da esecuzione ad esecuzione, i programmi sono dinamici: si comportano diversamente in funzione dei dati in ingresso
- ‡ In C, l'istruzione condizionale viene realizzata tramite le parole chiave **if...else**, secondo la sintassi:





La selezione condizionale - 2

if (x)

```
    istruzione1; /* Eseguita solo se x è diversa da zero */  
    istruzione2; /* Eseguita sempre */
```

if (x)

```
    istruzione1; /* Eseguita solo se x è diversa da zero */  
else  
    istruzione2; /* Eseguita solo se x è uguale a zero */  
    istruzione3; /* Eseguita sempre */
```



La selezione condizionale - 3

- # L'istruzione **if** viene comunemente utilizzata per controllare la validità dei dati
- # **Problema**: Progettare un programma che stampi la radice quadrata di un numero (in input)

```
#include <stdio.h>
#include <math.h>
main()
{
    double num;

    printf("Introdurre un numero non negativo: ");
    scanf("%lf", &num); /* Lo specificatore %lf indica un tipo double */
    if (num<0)
        printf("Errore nel dato di ingresso: il numero è negativo.\n");
    else
        printf("La radice quadrata è: %f \n", sqrt(num));
    exit(0);
}
```

L'indentazione dopo le parole chiave **if** ed **else** ha il solo obiettivo di aumentare la leggibilità e non è funzionalmente significativa





Esempio 3: differenza di due interi

```
#include<stdio.h>
int main()
{
    int a, b, c;    //Dichiarazione delle variabili
    //Inserimento dati
    printf("Inserire primo intero: \n");
    scanf("%d", &a);
    printf("Inserire secondo intero: \n");
    scanf("%d", &b);
    if (a>b){      //Determinazione del numero maggiore
        c=a-b;
        //Stampa risultato
        printf("Risultato di %d - %d: %d\n", a, b, c);
    }
    else{
        c=b-a;
        //Stampa risultato
        printf("Risultato di %d - %d: %d\n", b, a, c);
    }
    return 0;
}
```

Costrutto di **selezione condizionale**

Il confronto fra espressioni - 1

- # L'espressione condizionale che compare in un'istruzione **if** è (di solito) costituita da un confronto tra due valori
- # **Attenzione**... alla differenza fra l'operatore di assegnamento "=" e l'operatore relazionale di uguaglianza "=="

```
if (j=5)  
do_something();
```

è sintatticamente corretta dato che tutte le espressioni hanno associato un valore; l'espressione "j=5" ha valore $5 \neq 0$ (vero) e `do_something()` viene eseguita sempre!

Gli operatori relazionali

<	minore
>	maggiore
<=	minore o uguale
>=	maggiore o uguale
==	uguale
!=	diverso



Il confronto fra espressioni - 2

- Il linguaggio C non rappresenta esplicitamente i tipi booleani, ma li realizza attraverso gli interi: zero equivale a **FALSE**; qualsiasi valore diverso da zero viene valutato **TRUE**

if (j)
istruzione;

Se j è diverso da zero, l'istruzione viene eseguita

- Esempio**

Espressione	Valore
$-1 < 0$	1
$0 > 1$	0
$0 == 0$	1
$1 != -1$	1
$1 >= -1$	1
$1 > 10$	0



Il confronto fra espressioni - 3

- # **Esempio:** Scrivere un programma che legge un carattere, stampandolo se è una lettera dell'alfabeto, ignorandolo in caso contrario

- # **Note:**
 - ◆ Esiste la funzione di libreria *isalpha()* che restituisce un valore diverso da 0 se l'argomento è una lettera
 - ◆ L'uso di una chiamata di funzione come espressione condizionale è comune in C



Il confronto fra espressioni - 4

```
#include <stdio.h>
#include <ctype.h> /* incluso per la funzione isalpha */

main()
{
    char ch;

    printf("Introdurre un carattere: ");
    scanf("%c", &ch);
    if (isalpha(ch))
        printf("%c", ch);
    else
        printf("%c non è un carattere alfabetico\n", ch);
    exit(0);
}
```



I blocchi di istruzioni - 1

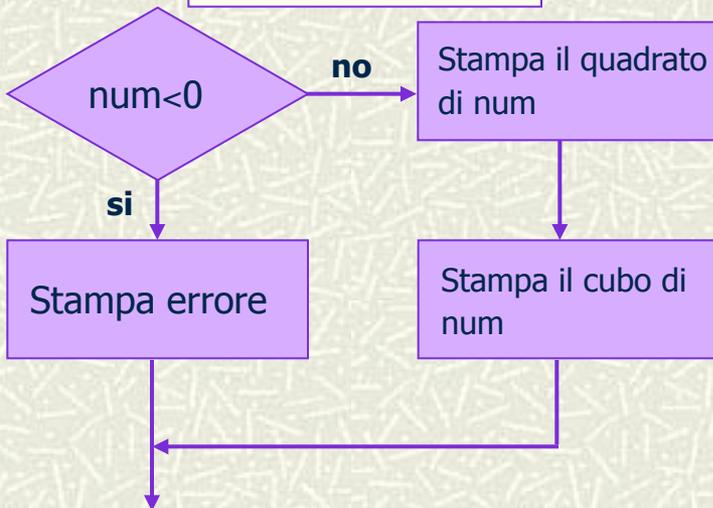
- ‡ Qualsiasi istruzione può essere sostituita da un ***blocco di istruzioni***
- ‡ Un blocco di istruzioni deve essere contenuto all'interno di parentesi graffe: il corpo di una funzione è un caso particolare di blocco di istruzioni
 - ⇒ Per eseguire in modo condizionale più di una singola istruzione è (necessario e) sufficiente racchiudere l'insieme di istruzioni in un blocco

```
#include <stdio.h>
main()
{
    double num;

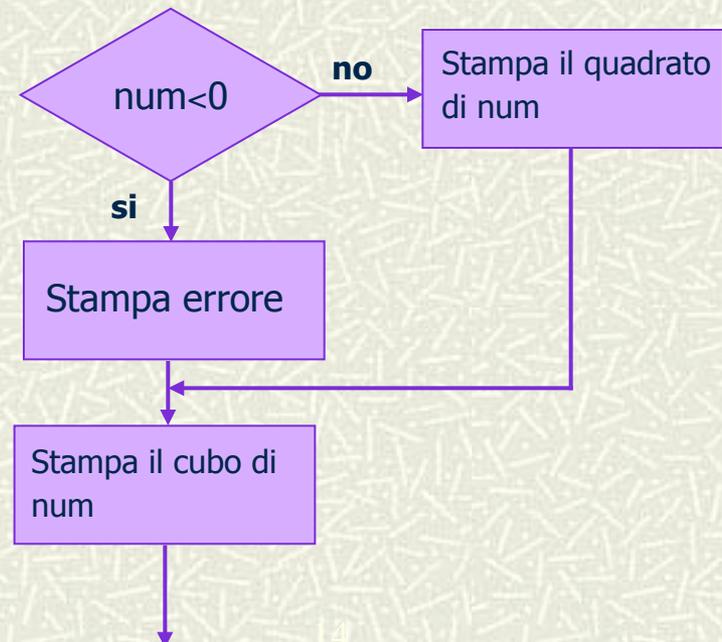
    printf("Introdurre un numero non negativo: ");
    scanf("%lf", &num);
    if (num<0)
        printf("Il numero immesso è negativo.\n");
    else
    {
        printf("Il quadrato di %lf è: %lf \n", num, num*num);
        printf("Il cubo di %lf è: %lf \n", num, num*num*num);
    }
    exit(0);
}
```

I blocchi di istruzioni - 2

```
if (num<0)
  print errore
else
{
  print quadrato
  print cubo
}
```



```
if (num<0)
  print errore
else
  print quadrato
  print cubo
```



Le parentesi graffe garantiscono la correttezza del flusso di controllo

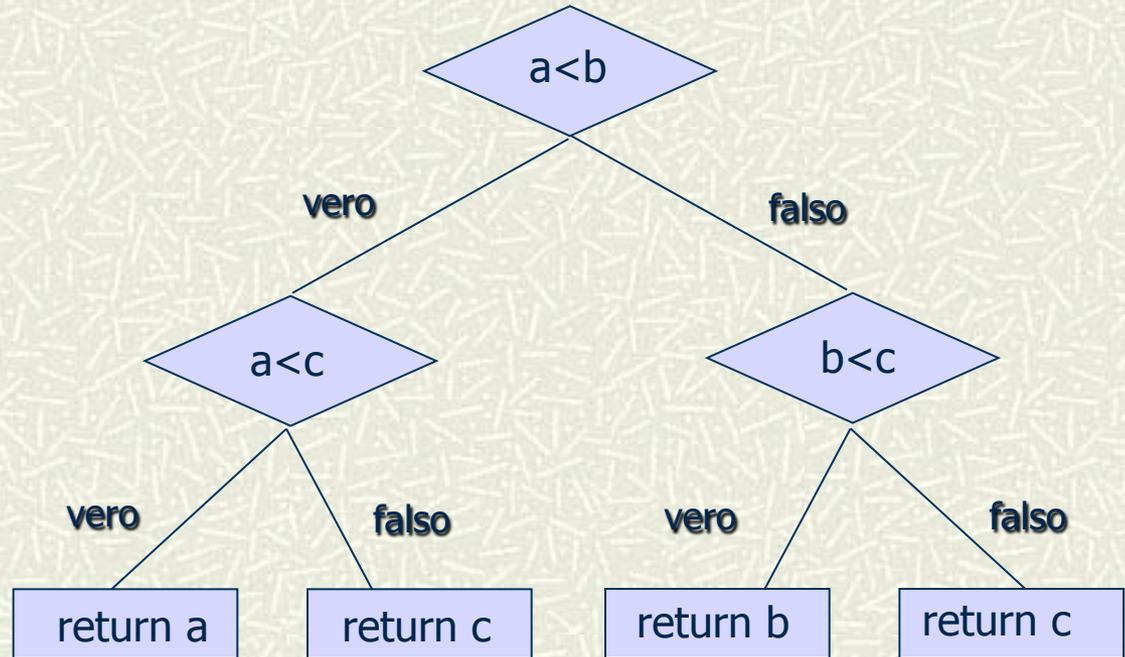


Istruzioni if innestate - 1

- ‡ Una singola istruzione **if** permette al programma di scegliere fra due alternative
- ‡ Talvolta è necessario specificare alternative successive: dopo aver preso la decisione_1, è necessario valutare la decisione_2, la decisione_3, etc.
- ‡ Questa tipologia di controllo del flusso richiede un costrutto **if innestato**
- ‡ **Esempio:** Realizzare una funzione che, dati tre interi, ne determina il minimo

Istruzioni if innestate - 2

```
int min(a, b, c)
int a, b, c;
{
  if (a<b)
    if (a<c)
      return a;
    else
      return c;
  else if (b<c)
    return b;
  else
    return c;
}
```





Istruzioni if innestate - 3

- # Nelle istruzioni **if** annidate sorge il problema di far corrispondere ad ogni clausola **else** l'opportuna istruzione **if**
- # **Regola:** *Una clausola **else** viene sempre associata all'istruzione **if** più vicina fra quelle precedenti*
 - ⇒ ad ogni istruzione **if** può corrispondere una sola clausola **else**
- # Per facilitare la programmazione, è opportuno indentare correttamente i vari **if**:
 - Una clausola **else** dovrebbe sempre essere posta allo stesso livello di indentazione dell'**if** associato
- # **Esempio:** soluzioni reali equazioni di 2° grado

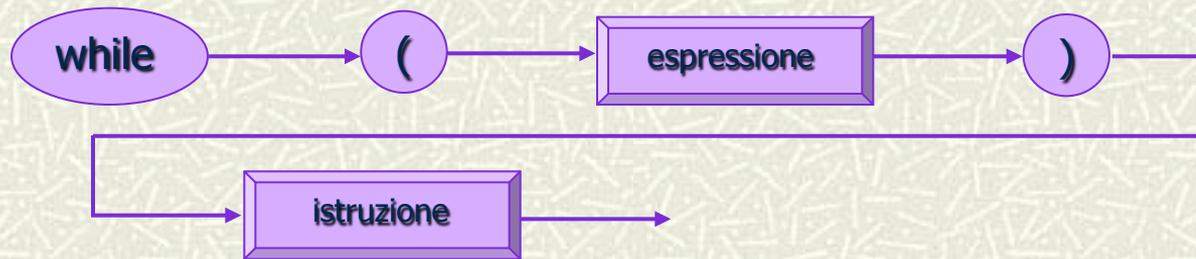


I cicli

- ‡ I **cicli**, o **iterazioni**, permettono l'esecuzione reiterata di un insieme di operazioni, fino al soddisfacimento di una condizione specificata
- ‡ Il linguaggio C offre tre istruzioni per la gestione dei cicli:
 - ◆ L'istruzione **while**
 - ◆ L'istruzione **do...while**
 - ◆ L'istruzione **for**

L'istruzione while - 1

‡ La sintassi dell'istruzione **while** è:

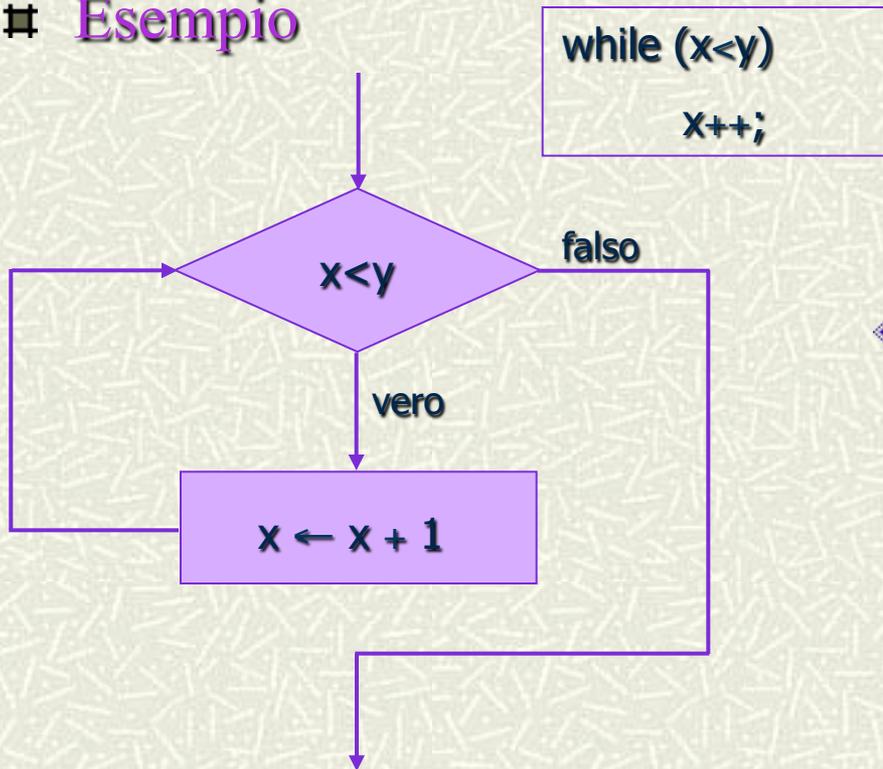


‡ Dal punto di vista semantico...

- ...si valuta **espressione**: se è diversa da zero (cioè vera) viene eseguita **istruzione** (che può essere composta e costituisce il **corpo del ciclo**) ed il controllo ritorna all'inizio del ciclo
- Si ripete l'esecuzione del ciclo fino al momento in cui **espressione** vale zero: il programma prosegue dal punto immediatamente successivo a **istruzione**

L'istruzione while - 2

▣ Esempio



◆ Nota:

L'operazione di incremento è così diffusa che il linguaggio C dispone di un apposito operatore di incremento, ++

$j = j + 1;$ ↔ $j++;$

L'istruzione while - 3

```
#include <stdio.h>
/* Calcola il fattoriale di n */
main()
{
    int n;
    unsigned long fatt;

    printf("Introdurre un intero positivo:\n");
    scanf("%d", &n);
    fatt = 1;
    while (n > 0)
    {
        fatt = fatt * n;
        n--;
    }
    printf("%d! = %ld\n", n, fatt);
    exit(0);
}
```

Calcolo del fattoriale





L'istruzione while - 4: somma di una sequenza di numeri

```
#include<stdio.h>
int main( )
{
    int somma, numero; //Dichiarazione variabili
    somma=0;           //Inizializzazione variabile
    printf("Inserire un numero (l'inserimento termina con 0): \n");
    scanf("%d", &numero);
    while(numero != 0)
    {
        somma = somma+numero;
        printf("Inserire un numero (l'inserimento termina con 0): \n");
        scanf("%d", &numero);
    }
    printf("Somma della sequenza inserita: %d\n", somma);
    return 0;
}
```

Ciclo while con controllo in testa ed iterazione per vero



L'istruzione while - 5

```
#include <stdio.h>
/* Conta il numero di spazi contenuti in una stringa
 * immessa da terminale
 */
main()
{
    int ch, num_of_spaces=0;
    printf("Introdurre una frase:\n");
    ch=getchar();
    while (ch != '\n')
    {
        if (ch == ' ')
            num_of_spaces ++;
        ch = getchar();
    }
    printf("Il numero di spazi è: %d.\n", num_of_spaces);
    exit(0);
}
```

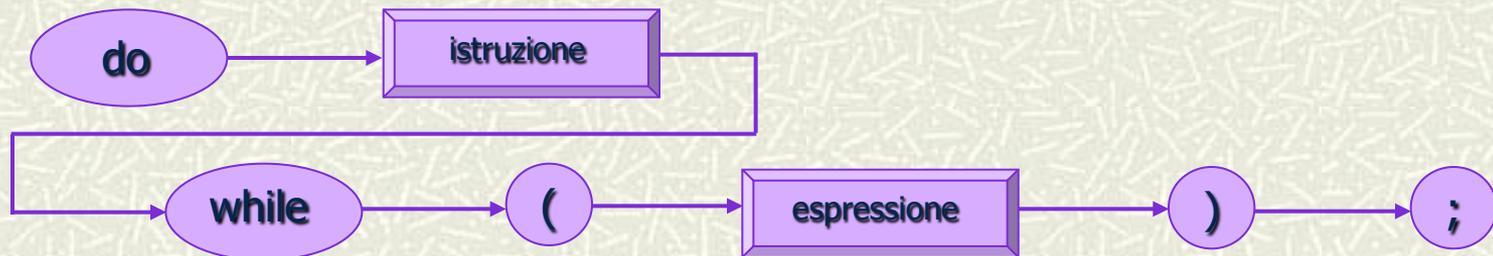
getchar() legge dati il cui tipo non è noto a priori e li fornisce sotto forma di **int**

Quando *getchar()* raggiunge la fine del file, viene restituito un valore speciale, detto **EOF**: un nome costante, definito in **stdio.h**, che solitamente vale -1

In questo caso, il carattere di fine stringa è **\n**

L'istruzione do...while - 1

- ‡ Nell'istruzione **while**, la condizione di fine ciclo è posta in testa all'iterazione: se la condizione è inizialmente falsa, il corpo del **while** non viene mai eseguito
- ‡ Esistono situazioni in cui è necessario che il corpo del ciclo sia eseguito almeno una volta: si utilizza l'istruzione **do...while**





L'istruzione do...while - 2

```
#include <stdio.h>
/* Conta il numero di spazi contenuti in una stringa
 * immessa da terminale
 */

main()
{
    int ch, num_of_spaces=0;
    printf("Introdurre una frase:\n");
    do
    {
        ch = getchar();
        if (ch == ' ')
            num_of_spaces ++;
    } while (ch != '\n');
    printf("Il numero di spazi è: %d.\n", num_of_spaces);
    exit(0);
}
```

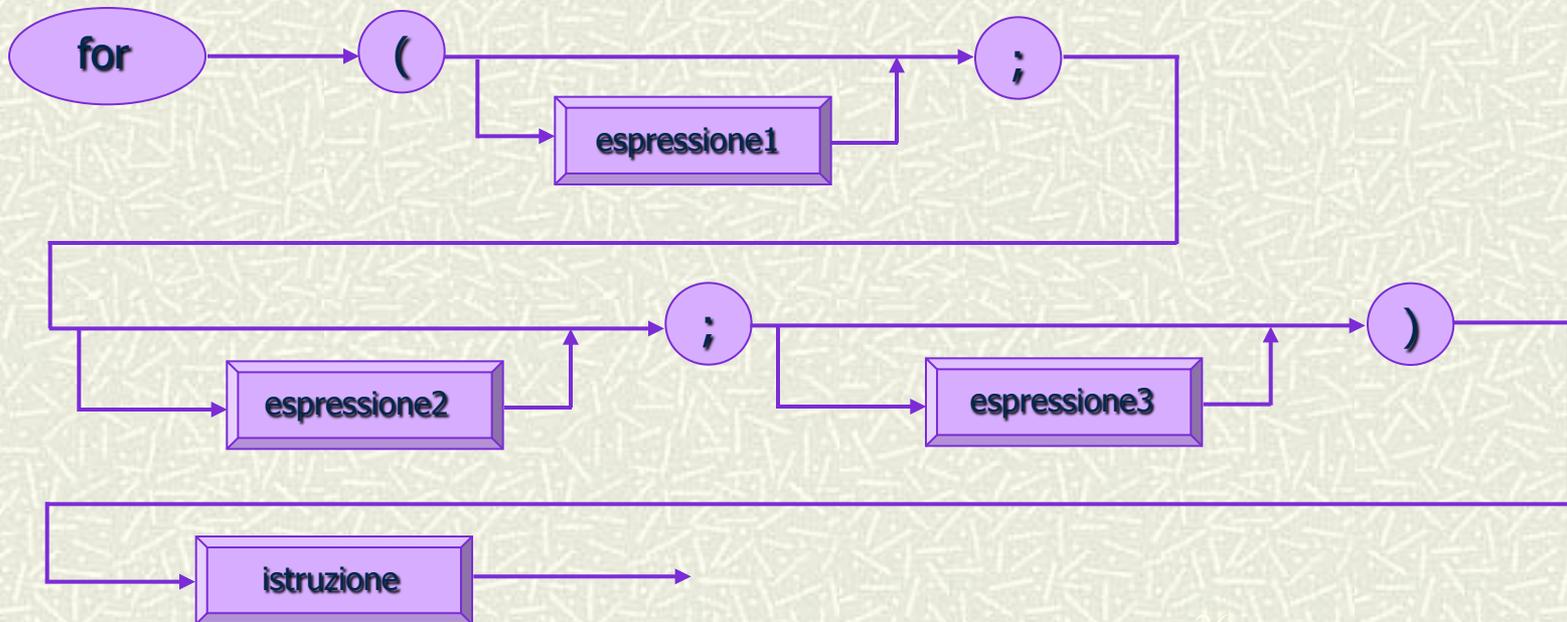
Il corpo del ciclo viene eseguito almeno una volta perché il controllo è in coda

Si noti che non è necessario prevedere l'inizializzazione di `ch`, poiché **do...while** garantisce l'analisi di almeno il primo carattere

Nota: malgrado il controllo in coda, l'iterazione è "per vero" come nel caso del costrutto **while**

L'istruzione for - 1

- ‡ L'istruzione **for** descrive una tipologia di ciclo in cui è necessario inizializzare una/più variabili prima di entrare nel ciclo, e modificare i loro valori ad ogni iterazione
- ‡ La sintassi dell'istruzione **for** è:





L'istruzione **for** - 2

- # L'istruzione **for** opera nel modo seguente:
 - Viene valutata **espressione1**: normalmente, è un'istruzione di assegnamento che inizializza una o più variabili
 - Viene valutata **espressione2**, che costituisce l'espressione condizionale dell'istruzione
 - Se **espressione2** è falsa si esce dall'istruzione **for** e il controllo passa all'istruzione immediatamente successiva nel programma; viceversa, se **espressione2** è vera, viene eseguita **istruzione**
 - Dopo che **istruzione** è stata eseguita, viene valutata **espressione3** e il controllo torna alla valutazione di **espressione2**
- # Si noti che **espressione1** viene valutata una sola volta, mentre **espressione2** ed **espressione3** vengono valutate ad ogni iterazione del ciclo



L'istruzione for - 3

Le istruzioni **while** e **for** a confronto:

```
expr1;  
while (expr2)  
{  
    istruzione;  
    expr3;  
}
```

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

Esempio: Calcolo del fattoriale

```
unsigned long factorial(val)  
int val;  
{  
    int j;  
    unsigned long fatt = 1;  
    for (j = 2; j <= val; j++)  
        fatt *= j;  
    return fatt;  
}
```

*= operatore di assegnamento aritmetico

fatt *= j; \longleftrightarrow fatt = fatt*j;





L'istruzione for - 4

- **Esempio:** Funzione che converte una stringa di cifre introdotte da tastiera in un numero intero

```
#include <stdio.h>
#include <ctype.h>
int make_int()
{
    int num=0, digit;
    digit = getchar();
    for(; isdigit(digit); digit=getchar())
    {
        num = num * 10;
        num = num + (digit - '0');
    }
    return num;
}
```

Funziona solo se i codici delle cifre da 0 a 9 sono contigui e crescenti: OK per il codice ASCII

In ASCII, il codice di '5' è 53 mentre quello di '0' è 48; se digit vale '5', l'espressione

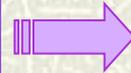
$$\text{digit} - \text{'0'} = 53 - 48 = 5$$



L'istruzione for - 5

- ‡ Una modalità alternativa prevede l'utilizzo del costrutto **while**:

```
#include <stdio.h>
#include <ctype.h>
int make_int1()
{
    int num=0, digit;
    while (isdigit(digit=getchar()))
    {
        num = num * 10;
        num = num + (digit - '0');
    }
    return num;
}
```



È la versione migliore: non richiede la doppia chiamata alla funzione di libreria di run-time *getchar()*



L'istruzione for - 6

Omissione delle espressioni

- ◆ **espressione1** e **espressione3** possono essere omesse, ma non in contemporanea: altrimenti si ottiene la stessa funzionalità di un ciclo **while**
- ◆ **espressione2** viene sempre inclusa

```
#include <stdio.h>

void newline_print(newline_num)
int newline_num;
{
    /* Stampa newline_num caratteri di ritorno a capo */
    for (; newline_num>0; newline_num--)
        printf("\n");
}
```



L'istruzione for - 7

Omissione del corpo del ciclo

- ◆ Si verifica quando l'elaborazione è effettuata direttamente all'interno delle espressioni
- ◆ È buona norma riservare al ";" una linea separata per renderlo più visibile

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

/* Legge i caratteri di spaziatura e li elimina */
void skip_spaces()
{
    int c;

    for (c=getchar(); isspace(c); c=getchar())
        ; /* Istruzione vuota */
    ungetc(c, stdin);
}
```

La funzione di libreria *isspace()* riconosce spazi, tabulazioni e ritorni a capo

La funzione *ungetc()* inserisce un carattere nel buffer di ingresso: **stdin** è un nome di macro definito in **stdio.h** che rappresenta il flusso di ingresso standard (di solito associato al terminale)

Errori comuni

- # **Uscita dal ciclo con un'iterazione di anticipo o di ritardo**
 - Dall'inglese *off-by-one-error*, è un errore molto comune, dovuto alla scelta non corretta dell'operatore relazionale
 - Non viene rilevato dal compilatore, né a run-time, perché il programma è sintatticamente corretto
- # **Inserimento del ";" dopo un'istruzione di controllo del flusso**
 - È sintatticamente corretto e quindi non viene notificato alcun errore di compilazione
 - Induce il compilatore ad eseguire un'istruzione vuota

```
if (j == 1);  
    j=0;
```

A j viene assegnato sempre il valore 0, indipendentemente dal suo valore iniziale

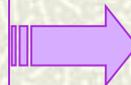




I cicli innestati

- # I costrutti iterativi possono essere innestati a qualunque livello di profondità
- # Il ciclo di livello n-esimo deve terminare la propria esecuzione prima che il ciclo al livello n-1 possa riprendere l'iterazione

Lo specificatore di formato `%5d` forza la funzione **`printf()`** a stampare 5 caratteri per ogni intero: se il numero richiede meno di 5 caratteri, viene preceduto da un numero appropriato di spazi bianchi



Esempio

```
#include <stdio.h>
/* Stampa una tavola pitagorica mediante cicli innestati */

main()
{
    int j, k;
    printf("  1  2  3  4  5  6  7  8  9 10\n");
    printf("  -----\n");

    for (j=1; j<=10; j++)
    {
        printf("%5d|", j);
        for (k=1; k<=10; k++)
            printf("%5d", j*k);
        printf("\n");
    }
    exit(0);
}
```



Stampa della Tavola Pitagorica

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100



L'istruzione switch - 1

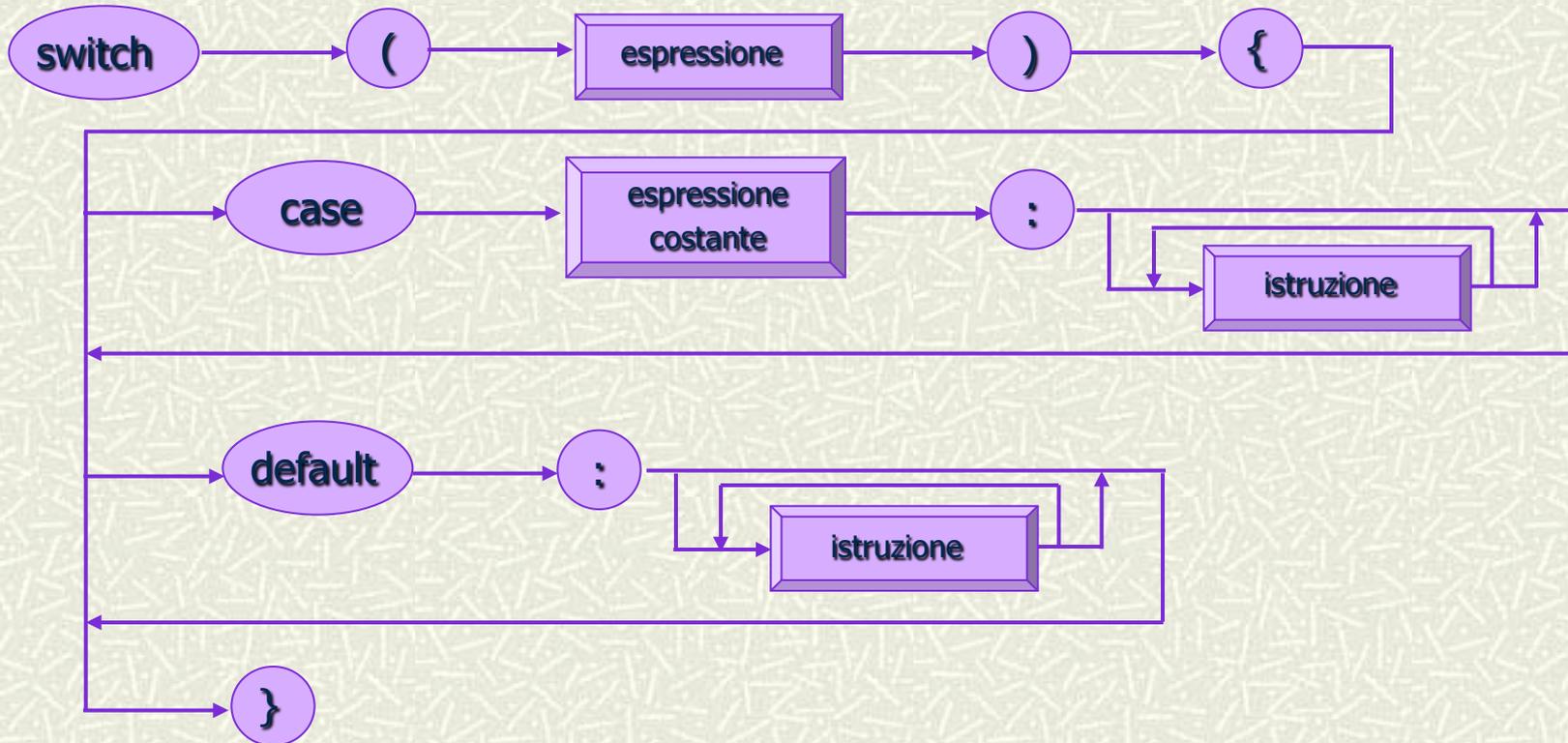
- # In presenza di cammini multipli all'interno di un programma, le diramazioni **if...else** possono complicare la comprensione del codice
- # L'istruzione **switch** consente di specificare un numero illimitato di cammini di esecuzione in dipendenza dal valore di un'espressione

```
int switch_example0(input_arg)
char input_arg;
{
    if (input_arg == 'A')
        return 1;
    if (input_arg == 'B')
        return 2;
    if (input_arg == 'C')
        return 3;
    if (input_arg == 'D')
        return 4;
    else
        return -1;
}
```

```
int switch_example(input_arg)
char input_arg;
{
    switch(input_arg)
    {
        case 'A': return 1;
        case 'B': return 2;
        case 'C': return 3;
        case 'D': return 4;
        default : return -1;
    }
}
```

L'istruzione switch - 2

‡ La sintassi dell'istruzione **switch** è:





L'istruzione switch - 3

- ‡ La semantica dell'istruzione **switch** impone che...
 - ...venga valutata l'espressione **intera**: se corrisponde ad uno dei valori **case**, il flusso di esecuzione viene trasferito alla prima istruzione successiva al **case** nel programma
 - ...se non c'è corrispondenza tra l'espressione **switch** ed alcun valore **case**, allora il flusso di esecuzione viene trasferito all'istruzione del ramo default (non obbligatorio)
- ‡ Non possono esistere **case** con valori uguali
- ‡ Nell'istruzione **switch**, il flusso di esecuzione, a partire dal **case** selezionato, prosegue fino alla successiva istruzione di controllo del flusso o alla fine dell'istruzione **switch**
- ‡ L'istruzione **break** forza l'uscita dal costrutto **switch**, trasferendo il controllo alla prima istruzione che segue il costrutto



L'istruzione switch - 4

```
#include <stdio.h>
typedef enum {ERR_INPUT_VAL, ERR_OPERAND, ERR_OPERATOR, ERR_TYPE} ERROR_SET;
void print_error(error_code)
ERROR_SET error_code;

/* Stampa un messaggio di errore sulla base di un codice di errore */
{
    switch (error_code)
    {
        case ERR_INPUT_VAL:
            printf("Errore: input scorretto.\n");
            break;
        case ERR_OPERAND:
            printf("Errore: operando scorretto.\n");
            break;
        case ERR_OPERATOR:
            printf("Errore: operatore scorretto.\n");
            break;
        case ERR_TYPE:
            printf("Errore: dati non compatibili.\n");
            break;
        default: printf("Errore: codice non previsto %d \n", error_code);
            break;
    }
}
```



Potrebbe essere inserita in un file header, **err.h**, per poter essere inclusa da più file sorgente



L'istruzione switch - 5

- ‡ Talvolta è utile associare un medesimo gruppo di istruzioni a più di un ramo **case**
 - ⇒ è sufficiente scrivere più valori di scelta in modo consecutivo

```
int is_punc(arg)
char arg;
/* Restituisce 1 se l'argomento è un carattere di
 * interpunzione, 0 altrimenti
 */
{
    switch(arg)
    {
        case '\':
        case '\;':
        case '\:':
        case '\.':
        case '\!': return 1;
        default: return 0;
    }
}
```



L'istruzione switch - 6

```
#include "err.h" /* contiene la dichiarazione typedef di ERROR_SET */  
  
/* Calcola il valore di un'espressione di cui siano forniti i due operandi e l'operatore */  
  
double evaluate(op1, operator, op2)  
double op1, op2;  
char operator;  
{  
    extern void print_error();  
  
    switch(operator)  
    {  
        case '+': return op1 + op2;  
        case '-': return op1 - op2;  
        case '*': return op1 * op2;  
        case '/': return op1 / op2;  
        default: /* operatore scorretto */  
                print_error(ERR_OPERATOR);  
                exit(1);  
    }  
}
```



Le istruzioni **break** e **continue** - 1

- # L'uso dell'istruzione **break** all'interno del costrutto **switch** evita che il controllo del programma passi da un **case** al successivo
- # L'uso di **break** all'interno di una struttura iterativa provoca l'uscita dal ciclo e l'esecuzione della prima istruzione che segue il corpo del ciclo
- # Le istruzioni **break** devono essere evitate, quando possibile: producono salti (incondizionati) e rendono il programma di difficile lettura e comprensione
- # È possibile scrivere cicli con modalità alternative, che non prevedono l'uso di **break**; **break** è invece "insostituibile" nel caso di **switch**

```
for (cnt=0; cnt<50; cnt++)
{
    c = getchar();
    if (c == "\n")
        break;
    else
        /* elaborazione del carattere */
        ... ..
}
/* L'istruzione break fa riprendere
 * l'esecuzione da questo punto
 */
```



Le istruzioni break e continue - 2

- ‡ L'istruzione **continue** consente di ritornare anticipatamente il controllo all'inizio di un ciclo
⇒ utile quando è necessario evitare di eseguire parte del corpo del ciclo
- ‡ L'istruzione **continue** deve essere evitata, quando possibile, perché, come **break**, altera il flusso di controllo naturale (attraverso un salto incondizionato)

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

int make_int2()
{
    int num=0, digit;
    while ((digit=getchar()) != '\n')
    {
        if (isdigit(digit) == 0)
            continue;
        num = num * 10;
        num = num + (digit - '0');
    }
    return num;
}
```



L'istruzione goto

- ✦ L'istruzione **goto** ha lo scopo di trasferire il controllo del flusso in un punto particolare, identificato da un' **etichetta** o **label**
- ✦ La label è un nome seguito da “:” e deve essere contenuta nella stessa funzione che contiene il **goto** che vi fa riferimento
- ✦ Il **goto** deve essere usato solo quando il codice risultante guadagna in efficienza senza perdere troppo in leggibilità

```
... ..  
goto label1;  
  
... ..  
label1: /* istruzione dalla quale si riprende  
        * l'esecuzione dopo il goto  
        */
```

I cicli infiniti - 1

- # Un ciclo infinito non contiene una condizione di terminazione, o tale condizione non viene mai verificata
- # Nella maggior parte dei casi, un ciclo infinito è il frutto di un errore di programmazione
- # **Esempio:**

```
for (j=0; j<10; j++)  
{  
    ... ..  
    j = 1;  
}
```

non termina perché a j viene riassegnato il valore 1 ad ogni iterazione



I cicli infiniti - 2

- Si possono ottenere cicli infiniti con le istruzioni:

```
while(1)
    istruzione;
```

```
for(;;)
    istruzione;
```

Programma calcolatrice

```
#include <stdio.h>

main()
{
    extern double evaluate();
    double op1, op2;
    char operator;
    while (1)
    {
        printf("Introdurre <number> <op> <number> <newline>: ");
        scanf("%lf %c %lf", &op1, &operator, &op2);
        printf("%lf", evaluate(op1,operator,op2));
    }
}
```

L'esecuzione procede fino a quando viene interrotta con la combinazione di tasti CTRL-C



Esempio 1: Numeri pari e dispari

- # **Problema:** Scrivere un programma per stabilire se un numero intero è pari o dispari
- # **Suggerimento:** l'operatore modulo, $x \% y$, restituisce il resto della divisione intera di x per y

```
#include <stdio.h>
main()
{
    int num, resto;
    printf("Immetti un numero intero: ");
    scanf("%d", &num);
    resto = num % 2;
    if (resto == 0)
        printf("Il numero è pari");
    else
        printf("Il numero è dispari");
}
```

Esempio 2: Conteggio di caratteri

```
#include <stdio.h>
main()
{
    int c;
    long int nc;

    c = getchar();
    nc = 1;
    while (c != EOF)
    {
        c = getchar();
        nc = nc + 1;
    }
    printf("Totale: %l\n", nc);
}
```



```
#include <stdio.h>
main()
{
    long int nc = 1;
    while (getchar() != EOF)
        nc++;
    printf("Totale: %l\n", nc);
}
```



Esempio 3: Conteggio di linee

```
#include <stdio.h>

main()
{
    int c;
    long int nr = 0;
    while ((c = getchar()) != EOF)
    {
        if (c == '\n')
            nr++;
    }
    printf("Linee: %d\n", nr);
}
```



Esempio 4: Da stringhe a floating

```
#include <stdio.h>
#include <ctype.h>
#define DECIMAL_POINT '.'

double parse_num()
{
    int c, j, digit_count=0;
    double value=0, fractional_digit;

    while (isdigit(c=getchar()))
    {
        value = value * 10;
        value = value + (c - '0');
    }
    /* Quando c non è una cifra si controlla se è un punto decimale */
    if (c == DECIMAL_POINT) /* se vera, si legge la parte frazionaria */
        while (isdigit(c=getchar()))
        {
            digit_count++;
            fractional_digit = c - '0';
            for (j=0; j<digit_count; j++)
                fractional_digit /= 10;
            value = value + fractional_digit;
        }
    ungetc(c, stdin);
    return value;
}
```



Esempio 5: Numeri di Fibonacci

```
#include <stdio.h>
main()
{
    int ultimo, penultimo, i, f;
    penultimo = 0;
    printf("%d\n", penultimo);
    ultimo = 1;
    printf("%d\n", ultimo);
    for (i=2; i<=20; i=i+1)
    {
        f = ultimo + penultimo;
        printf("%d\n", f);
        penultimo = ultimo;
        ultimo = f;
    }
}
```



```
#include <stdio.h>
main()
{
    int ultimo=1, penultimo=0, i, f;
    printf("0\n1\n");
    for (i=2; i<=20; i++)
    {
        f = ultimo + penultimo;
        penultimo = ultimo;
        printf("%d\n", ultimo = f);
    }
}
```