Sistemi Operativi

Unità 3: Programmazione in C

Controllo del flusso e cicli

Martino Trevisan
Università di Trieste
Dipartimento di Ingegneria e Architettura

Argomenti

1. Controllo del flusso
2. Cicli

Controllo del flusso

Controllo del flusso

Definizione

La possibilità di eseguire set alternativi di istruzioni a seconda del verificarsi di una condizione.

- Alla base di quasi ogni programma.
- E' necessario definire una condizione, ovvero una **espressione booleana** che può essere vera (`true` = o false (`false`)
- Vedremo:
  - Operatori di confronto: `==` , `!=` , `<` , `>` , `<` , `>=` , `<=`
  - Operatori di booleani: `&&` , `||` , `!`

Controllo del flusso

Istruzione `if-else`

```
if ( condizione )
{
    A; // Ramo Vero
}
else
{
    B; // Ramo Falso
}
```

Concettualmente identico ai costrutti `if` in Python o Java.

Controllo del flusso

Istruzione `if-else`

Possibile avere più possibili rami.

```
if ( condizione1 )
{
    A; // Eseguito se condizione1
}
else if (condizione2)
{
    B; // Eseguito se condizione2
}
else
{
    C; // Eseguito altrimenti
}
```

Controllo del flusso

Istruzione `if-else`

Esercizio: si scriva un programma che legge da tastiera un intero e scrive se esso è positivo o negativo

```c
#include <stdio.h>

int main(void)
{
    int a;
    printf("inserisci un numero: ");
    scanf("%d", &a);

    if (a>0){
        printf("%d e' positivo\n", a);
    }
    else
    {
        printf("%d e' negativo\n", a);
    }
}
```

Controllo del flusso

Istruzione `if-else`

Osservazioni:

- E' possibile omettere il ramo `else`:

```
if ( condizione )
{
    A; // Ramo Vero
}
```

Controllo del flusso

Istruzione `if-else`

- Se un ramo è composto da una sola istruzione, è possibile omettere le `{}`

```
if ( condizione )
    A;
else
{
    B;
}
```

oppure

```
if ( condizione )
    A;
```

Questo vale per **tutti i costrutti** in C: cicli `for` e `while`, ...

Controllo del flusso

Istruzione `if-else`

Esempio:

```
int a;
printf("inserisci un numero: ");
scanf("%d", &a);

if (a>0)
    printf("%d e' positivo\n", a);
else
    printf("%d e' negativo\n", a);
```

Errore comune: omettere le `{}` quando il blocco ha più di una istruzione!

Controllo del flusso

Operatori di confronto

Equivalenti a quelli di Java o Python

- Uguaglianza: `a == b`
- Differenza: `a != b`
- Maggiore: `a > b`
- Maggiore o uguale: `a >= b`
- Minore: `a < b`
- Minore o uguale: `a <= b`

Errore comune: confondere operatore di assegnazione `=` con quello di uguaglianza `==`

Controllo del flusso

Operatori booleani

Gli operatori di confronto permettono di denifinire condizioni semplici.

Spesso è necessario combinare condizioni semplici.

Esempio: un valore è compreso in un intervallo?

`a>10` **e** `a<20`

Per combinare condizioni semplici si usano gli operatori booleani.

- AND: `(cond1) && (cond2)`
- OR: `(cond1) || (cond2)`
- NOT: `!(cond1)`

Controllo del flusso

Operatori booleani

Precedenza degli operatori nelle condizioni. Ordine di priorità:

- Operatori di confronto
- Operatore `!`
- Operatore `&&`
- Operatore `||`

Esempi:

```
if ( (a>10) && (b>10) )
/* Equivale a */
if ( a>10 && b>10 )
```

```
if ( !(a>10) || (b>10) )
/* Equivale a */
if ( !a>10 || b>10 )
```

Controllo del flusso

Errore comune

Confondere operatore di AND booleano `&&` con l'operatore di *bitwise* AND `&`.

Annidamento

E' possibile annidare istruzioni `if else` per creare ramificazioni complesse.

```
if ( condizione1 )
    if ( condizione2 ){
        A;
    }
    else    {
        B;
    }
else{
    C;
}
```

Controllo del flusso

Si scriva un programma che risolve un'equazione di primo grado.

```c
#include <stdio.h>
int main()
{
    float a, b ; /* coefficienti a e b */
    float x ; /* valore di x che risolve l'equazione */
    printf("Risoluzione di un'equazinoe di primo grado\n");
    printf("Equazione nella forma: ax + b = 0\n") ;
    /* LEGGI a e b */
    printf("Immetti coefficiente a: ");
    scanf("%f", &a);
    printf("Immetti coefficiente b: ");
    scanf("%f", &b) ;

    /* x VIENE CALCOLATO COME x=-b/a. SI DEVONO VERIFICARE I VALORI DI a E b */
    if( a != 0 ) {
        x = - b / a;
        printf("La soluzione e' x = %f\n", x);
    } else { /* CASO a==0 */
        if( b==0 ) {
            printf("Equazione indeterminata (ammette infinite soluzioni)\n");
        } else {
            printf("Equazione impossibile (non ammette soluzioni)\n");
        }
   }
}
```

Controllo del flusso

Istruzione `switch`

Semplifica il codice in caso di scelta in base al valore di una espressione.

Sintassi:

```
switch (espressione)
{
    case v1:
        A;
    break;
    case v2:
        B;
    break;
    default:
        C;
}
```

Controllo del flusso

Istruzione `switch`

Osservazioni:

- `espressione` può essere una variabile o un'espressione
- `v1`, `v2` devono essere una costante. Non possono essere una variabile.
- Necessario sempre delimitare ogni caso con `case` e `break`
  - **Errore comune**: dimenticare il `break`
- `default` si comporta come `else` nel costrutto `if`. E' opzionale.

Controllo del flusso

Istruzione `switch`

Osservazioni:

- E' un operazione particolare: non utilizza `{}` ma `case` e `break`.
  - Relitto di istruzioni con `goto`.
- Non utilizzare se non serve strettamente.
  - Usata tipicamente per migliore performance rispetto a `if` quando ho tanti casi.

Controllo del flusso

Operatore ternario

Permette di scrivere in maniera concisa un'espressione `if then else`.

Sintassi:

```
var = condizione ? espressione1 : espressione2;
```

Equivale a:

```
if(condizione)
    var = espressione1;
else
    var = espressione2;
```

Controllo del flusso

Operatore ternario

Limitazione

`espressione1` e `espressione2` possono essere solo **espressioni**, non istruzioni!

L'operatore ternario può essere usato per fornire un'espressione in una istruzione

Esempi

Espressione:

```
c = (a < b) ? a : b; // Assegna a 'c' il minore tra 'a' e 'b'
```

Cicli

Cicli

Definizione

Hanno lo scopo di ripetere in maniera controllata un blocco di istruzioni.
Permettono di svolgere compiti ripetitivi senza duplicare il codice.
Solitamente organizzati in:

- Un blocco di istruzioni da eseguire ripetutamente.
- Una condizione che regola la fine del ciclo.

In C esistono due tipi di ciclo `while` (e variante `do-while`) e `for`.

Cicli

Ciclo `while`

Esegue finchè una condizione è vera.

```
while ( C )
{
    A;
}
```

Comportamento:

1. Viene valutata C.
2. Se C è falsa, salta il blocco di istruzioni
3. Se C è vera, esegue il blocco di istruzioni A e torna al punto 1

Cicli

Ciclo `while`

Esempio:

```
int i = 1;
while ( i <= 10 )
{
    printf("Numero = %d\n", i) ;
    i = i++; // Operatore di incremento
}
```

Risultato: stampa i numeri da 1 a 10 compresi.

Errore comune: sbagliare la condizione di terminazione e generare un ciclo infinito.

Cicli

Si scriva un programma che la media di un numero di `float` specificato dall'utente.

```c
#include <stdio.h>
int main()
{
    int i=0, n;
    float dato;
    float somma = 0.0;

    printf("Introduci n: "); /* Leggi n */
    scanf("%d", &n) ;
    if( n>0 )
    {
        while(i < n )  /* Esegui n volte */
        {
            printf("Valore %d: ", i+1) ;
            scanf("%f", &dato) ;
            somma = somma + dato ; /* Accumula la somma */
            i = i + 1 ; /* Contatore */
        }
        printf("Risultato: %f\n", somma/n) ;
    }
    else
        printf("Non ci sono dati da inserire\n");
}
```

Cicli

Ciclo `for`

Rende più facile eseguire un blocco N volte.

- E' possibile anche col ciclo `while` ma è più prono a errori, siccome è necessario:
  i. inizializzare un contatore
  ii. impostare la condizione del `while`
  iii. incrementare il contatore a ogni operazione

Il ciclo `for` sistematizza queste operazioni

Cicli

Ciclo `for`

Sintassi

```
for ( I; C; A )
{
    B;
}
```

- `I` è l'**istruzione** di inizializzazione
- `C` è la **condizione** di terminazione
- `A` è l'**istruzione** di aggiornamento

Cicli

Ciclo `for`

Esempio:

```
int i ;
for ( i=0; i<10; i=i++ )
{
    printf("Numero = %d\n", i) ;
}
```

equivale a:

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

Cicli

Ciclo `for`

E' semplice annidare cicli `for`, ad esempio per iterare su una tabella o matrice.

```
for( i=0; i<N; i++ )
    {
    for( j=0; j<N; j++ )
    {
        printf("i=%d - j=%d\n", i, j);
    }
}
```

Cicli

Esercizio: si crei un programma che determina su un numero inserito dall'utente è primo.

```c
#include <stdio.h>
int main()
{
    int n, i;
    int primo=1;

    printf("inserisci il numero: ");
    scanf("%d", &n);

    for (i=2; i<n; i++)
        if(n%i==0)
            primo=0;

    if ( primo == 0)
        printf("Il numero %d non è primo\n", n);
    else
        printf("Il numero %d è primo\n", n);

}
```

Domanda: il numero $2\,147\,483\,647$ è primo? Quanto ci mette a calcolare?

Cicli

Ciclo `for`

Osservazioni

- Come nell'esempio precedente, se il blocco ha una sola istruzione, si possono omettere le `{}`
- Possibile annidare blocchi di una istruzione in più costrutti di flusso.
- Esempio:

```
for (i=2; i<n; i++)
  if(n%i==0)
      primo=0;
```

Cicli

Istruzioni speciali

All'interno dei cicli `for` e `while` è possibile usare le seguenti istruzioni speciali.

- `break`: termina il ciclo immediatamente, passando alle istruzioni seguenti al ciclo.
- `continue`: passa immediatamente alla iterazione successiva senza eseguire le rimanenti istruzioni del blocco.

Cicli

Istruzioni speciali

Esercizio: quante volte viene invocata la `printf`?

```
for (i=0; i<10;i++){
    printf("Iterazione\n");
    if (i==3)
        break;
}
```

Esercizio: quante volte viene invocata la `printf`?

```
for (i=0; i<10;i++){
    if (i<3)
        continue;
    printf("Iterazione\n");
    if (i>5)
        break;
}
```