Fondamenti di Informatica L-A

Esercitazione 3

Espressioni: precedenza e associatività degli operatori.
Programmazione strutturata: blocchi, condizioni, cicli.

Istruzioni `if`, `switch`, `do...while`, `while`, `for`.

Paolo Torroni

Università degli Studi di Bologna
Laurea in Ingegneria Elettronica e delle Telecomunicazioni

Anno Accademico 2007/2008

△ 1. Precedenza e associatività degli operatori ▽

- Si consideri la seguente tabella degli operatori in C:

Precedenza	Operatori	Associatività
1	()	a sinistra
2	! ++ --	a destra
3	* / %	a sinistra
4	+ -	a sinistra
6	< <= > >=	a sinistra
7	== !=	a sinistra
11	&&	a sinistra
12	||	a sinistra
14	= += -= *=	a destra
15	,	a sinistra

- Siano date le segnuenti definizioni di variabile:

```
typedef unsigned short int _us;
int A=1; _us B=20; char C='k'; float E=11.5;
```

- Come vengono valutate le seguenti espressioni?
  1. **Expr1**: (int) E % B / 2 != C $\leq$ (A = B * C || B % 2)
  2. **Expr2**: C -= (++B / printf("%c%c", C, C)) / C + A++
- Si verifichi con il debugger la correttezza delle risposte date.

△ 2. Istruzione condizionale `if` ▽

- Si consideri il seguente programma C:

```
#include <stdio.h>
main() {
    char A1; int N=10;
    printf( "Please type your command (up/down)...\t" );
    scanf( "%c", &A1 );
    if( A1=='u' ) N++;
    else if( A1=='d' ) N--;
    printf( "N is now %d\n", N );
}
```

- Cosa fa il programma?
  - Si provi a seguire l'esecuzione con il debugger
- Si modifichi il codice in modo che:
  1. Il programma non tenga conto delle maiuscole/minuscole nell'input dell'utente (cioè funzioni con 'u', 'U', 'd', e 'D').
  2. L'utente sia in grado di partire da un numero N a piacere.

NOTA: Se si dovessero verificare malfunzionamenti della `scanf` *in lettura di un carattere, si utilizzi l'istruzione* `fflush(stdin);` *esemplificata nel punto 5.*

⋘ Istruzione condizionale `switch` ⋙

- L' istruzione `switch` è semplicemente un modo per effettuare vari `if` in cascata usando una notazione più compatta.

```
#include <stdio.h>
main() {
    int N=2; // provare diversi valori!
    switch( N ) {
    case 1: printf( "N vale 1\n" ); break;
    case 2: case 3: printf( "N vale 2 o 3\n" );
    case 5: printf( "N vale 2, 3, o 5\n" ); break;
    default: printf( "N non vale 1, 2, 3, o 5\n" );
    }
}
```

- La condizione dello `switch` deve essere un' espressione che restituisce **un intero**
- "`case`" seleziona il punto di ingresso, a seconda del valore di `N`;
- "`default:`" seleziona un punto di ingresso di default, se non è stato possibile entrare prima;
- "`break;`" viene usato per uscire dallo `switch`.

△ 3. Istruzione condizionale `switch` ▽

- Si consideri il programma C del punto 2:

```
#include <stdio.h>
main() {
    char A1; int N=10;
    printf( "Please type your command (up/down)...\t" );
    scanf( "%c", &A1 );
    if( A1=='u' ) N++;
    else if( A1=='d' ) N--;
    printf( "N is now %d\n", N );
}
```

- Si modifichi il codice, utilizzando l'istruzione `switch` al posto delle istruzioni `if` in cascata.
  - Come prima, il programma deve riconoscere '`u`', '`U`', '`d`', e '`D`'
  - Si aggiunga un ulteriore comando, '`m`' (oppure '`M`') che
    1. legga da input un altro numero, M;
    2. aggiorni N con il valore $N \times M$.

△ 4. Cicli `while` ▽

- Si consideri il seguente programma C:

```
#include <stdio.h>
main() {
    int N=18, M=15;
    printf( "mcm(%d,%d): ", N, M );
    while ( N!=M )
        if( N>M ) N -= M;
        else M -= N;
    printf( "%d\n", N );
}
```

- Cosa fa il programma?
  - Si facciano varie prove utilizzando diversi valori di M ed N.
- Si apportino le seguenti modifiche al programma:
  1. Si faccia in modo che, al termine dell'esecuzione, l'utente possa decidere di ripetere l'esecuzione con nuovi valori di input M, N.
  2. Si faccia il modo che il programma visualizzi il numero di operazioni di confronto (`N!=M` e `N>M`) effettuate.

⋘ Altri cicli: `do...while` e `for` ⋙

```
#include <stdio.h>
main() {
    int i, N=10; char C;

    // ciclo while
    i=0; C='a';
    while( i<N ) {
     printf( "%c", a++ ); i++;
    }

    // ciclo do..while
    i=0; C='a';
    do{
     printf( "%c", a++ ); i++;
    } while (i<N);

    // ciclo for
    for( i=0, C='a'; i<N; i++ )
     printf( "%c", a++ );
}
```

- Esistono altre due istruzioni in C che si possono usare in alternativa a `while`:
  - il ciclo `do...while`, che prima esegue la prima istruzione, poi verifica la condizione per proseguire;
  - il ciclo `for`, che usa una notazione compatta che indica: (**inizializzazione** ; **condizione** per iniziare/proseguire ; istruzione/i da eseguire **alla fine** di **ogni** ciclo)
- Nell'esempio a sinistra, i tre cicli effettuano le stesse operazioni.
  - Quali sono?

△ 5. Cicli do...while ▽

- Si consideri il seguente programma C:

```
#include <stdio.h>
main() {
    char A1=0; int N=10;
    do {
        printf( "Please type a command (up/down)...\t" );
        fflush( stdin );
        scanf( "%c", &A1 );
    } while ( A1 != 'u' && A1 != 'd');
    if( A1=='u' ) N++;
    else if( A1=='d' ) N--;
    printf( "N is now %d\n", N );
}
```

- Cosa fa il programma?
  - Si provi a seguire l'esecuzione con il debugger
- Si modifichi il codice in modo da includere il comando 'm' descritto al punto 3.

NOTA: `fflush(stdin);` *serve per evitare problemi nella lettura di caratteri.*

△ 6. Cicli `for` ▽

- Si consideri il seguente programma C:

```
#include <stdio.h>
main() {
    int N=18, M=15, i, A=1;
    printf( "mcm(%d,%d): ", N, M );
    for( i=2; i<N; i++ )
        if( !(N%i) && !(M%i) )
            A=i;
    printf( "%d\n", A );
}
```

- Cosa fa il programma?
  - Si provi a seguire l'esecuzione con il debugger
  - Si facciano varie prove utilizzando diversi valori di M ed N.
- Si apporti la modifica 1. specificata al punto 4.
- Si faccia il modo che il programma visualizzi il numero di operazioni modulo (`N%i` e `M%i`) effettuate.