

Approfondimenti sul ciclo while

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2012/2013

Introduzione

In questa lezione. . .

- ▶ Mostriamo semplici esempi di uso dei cicli
- ▶ Ciascuno di questi esempi mostra un metodo d'uso dei cicli che va capito a fondo e memorizzato
- ▶ Altri e nuovi problemi che vedremo in futuro potranno essere risolti riconducendosi ai questi *pattern*



Section 2

Pattern importanti



Esempio 1: contatore

- Leggere una sequenza di numeri di standard input che termini con lo 0 e contare quanti di questi (escluso lo 0) sono pari

```
int contatore;  
int input;  
  
int main() {  
    scanf("%d", &input);  
  
    contatore = 0; ← INIZIALIZZAZIONE DEL  
                  CONTATORE  
  
    while (input != 0) { ← CONDIZIONE  
        if (input % 2 == 0) {  
            contatore = contatore + 1; ← AGGIORNAMENTO DEL  
                                      CONTATORE  
        }  
        scanf("%d", &input);  
    }  
  
    printf("\nIl numero di numeri pari inserito e' %d", contatore  
}
```



Esempio 2: accumulatore

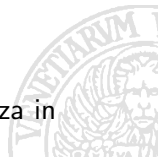
- Leggere una sequenza di numeri di standard input che termini con lo 0 e calcolarne la somma

```
int a;  
int acc; /*accumulatore*/  
int main(){  
    acc = 0;  
    scanf("%d", &a);  
    while (a != 0) {  
        acc = acc + a;  
        scanf("%d", &a);  
    }  
    printf("Somma=%d", a);  
    return 0;  
}
```

Inizializzazione
dell'accumulatore

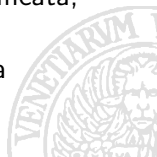
Aggiornamento
dell'accumulatore

- Applicare il pattern per calcolare la media della sequenza in input (che termina con lo zero).



Esempio 3 (proprietà universale)

- ▶ Leggere una sequenza di interi in input (che termina con lo 0) e stampare in output se sono **tutti** positivi (eccetto l'ultimo) oppure se ci sono anche dei numeri negativi
- ▶ Questo pattern è caratterizzato dal verificarsi di una proposizione su **tutti** gli elementi di un insieme
 - ▶ Siano $i_1, i_2, \dots, i_N, 0$ la sequenza in input
 - ▶ Dobbiamo verificare la proposizione $\forall j \in [1, N], i_j > 0$
- ▶ **Soluzione:** assumiamo che la proprietà richiesta sia verificata, e cerchiamo un controesempio, cioè un elemento della sequenza in input che non soddisfi la proprietà richiesta



Soluzione

```
int a;
int flag; /*stato della condizione*/
int main(){
    flag = 1;
    scanf("%d", &a);
    while (a != 0) {
        if (a < 0) {
            flag = 0;
        }
        scanf("%d", &a);
    }
    if (flag) {
        printf("Tutti i numeri sono poisitivi");
    }
    else printf("Almeno un numero è negativo");
    return 0;
}
```

Assumiamo la proprietà soddisfatta (flag con valore true)


Trovato un controesempio



Soluzione **ERRATA**

```
int a;
int flag; /*stato della condizione*/
int main(){
    flag = 1;
    scanf("%d", &a);
    while (a != 0) {
        if (a < 0) {
            flag = 0;
        }
        else {
            flag = 1;
        }
        scanf("%d", &a);
    }
    if (flag) {
        printf("Tutti positivi");
    }
    else printf("Almeno un numero è negativo");
    return 0;
}
```

Questo è l'errore, perchè?



Esempio 3 (proprietà esistenziale)

- ▶ Leggere una sequenza di interi in input (che termina con lo 0) e stampare in output se almeno uno è positivo
- ▶ Questo pattern è caratterizzato dal verificarsi di una proposizione su **almeno un** elemento di un insieme
 - ▶ Siano $i_1, i_2, \dots, i_N, 0$ la sequenza in input
 - ▶ Dobbiamo verificare la proposizione $\exists j \in [1, N], i_j > 0$
- ▶ **Soluzione:** assumiamo che la proprietà richiesta **non** sia verificata, e cerchiamo quell'elemento della sequenza in input che la soddisfi



Soluzione

```
int a;  
int flag; /*stato della condizione*/  
int main(){  
    flag = 0;  
    scanf("%d", &a);  
    while (a != 0) {  
        if (a > 0) {  
            flag = 1;  
        }  
        scanf("%d", &a);  
    }  
    if (flag) {  
        printf("Almeno un numero è positivo");  
    }  
    else printf("Tutti i numeri sono negativi");  
    return 0;  
}
```

Assumiamo la proprietà
non soddisfatta (flag con valore
false)

Trovato un esempio



Esempio 4 (trattare una sequenza)

- ▶ Leggere una sequenza di interi da standard input e fermarsi quando si leggono due valori consecutivi uguali.
- ▶ una soluzione errata:

```
int a;  
int b;  
int main(){  
    a = 0;  
    b = 1;  
    while (a != b) {  
        scanf("%d", &a);  
        scanf("%d", &b);  
    }  
    return 0;  
}
```

- ▶ Dare due esempi di sequenza per le quali il precedente programma non funziona



Soluzione

```
int a;  
int prec; /*penultimo valore letto*/  
int main(){  
    scanf("%d", &prec);  
    scanf("%d", %a);  
    while (a != prec) {  
        prec = a;  
        scanf("%d", &a);  
    }  
    return 0;  
}
```



Esempio 5 (trovare un confine)

- ▶ Un giocattolaio dispone di N mattoncini e vuole costruire una piramide. Ogni piano della piramide ha un mattoncino in meno di quello che sta sotto. L'ultimo piano ha un mattoncino. Siccome il giocattolaio comincia a costruire la piramide dalla base, vuole sapere quanti mattoncini deve usare per sfruttare al massimo gli N di cui dispone.
- ▶ Per esempio se $N = 22$, cominciare con 4 mattoncini non va bene perchè per fare la piramide gliene servono $4 + 3 + 2 + 1 = 10$, quindi gliene avanzano 12 che avrebbe potuto usare per fare almeno un altro piano di 5
- ▶ Per $N = 22$, cominciare con 7 non va bene perchè per completare la piramide servirebbe $7 + 6 + 5 + 4 + 3 + 2 + 1 = 28$ mattoncini
- ▶ La risposta corretta è 6 perchè $6 + 5 + 4 + 3 + 2 + 1 = 21$ e con 1 mattoncino non posso costruire un'altra riga

Prima soluzione: supero il confine e torno indietro

```
int matt;  
int base;  
  
int main() {  
    scanf("%d", &matt);  
  
    base=0;  
  
    while (matt >= 0) {  
        base=base+1;  
        matt = matt - base;  
    }  
  
    base = base - 1;   
    printf("La base ha %d mattoni", base);  
  
    reutrn 0;  
}
```

matt < 0 (ho usato troppi mattoni)


uso una base piu piccola



Seconda soluzione: mi fermo prima del confine guardando avanti

```
int matt;  
int base;  
  
int main() {  
    scanf("%d", &matt);  
  
    base=0;  
  
    while (matt-base-1 >= 0) {  
        base = base + 1;  
        matt = matt - base;  
    }  
  
    printf("La base ha %d mattoni", base);  
  
    reutrn 0;  
}
```

Guardo avanti



Section 3

Test di primalità



Decidere se un numero è primo: contare i divisori

- ▶ Un numero è primo se:
 - ▶ è maggiore di 1
 - ▶ è divisibile solo per 1 e per se stesso
- ▶ Idea *naïf* per il test di primalità di un numero n (**strategia**):
 - ▶ Se il numero è ≤ 1 allora non è primo
 - ▶ Altrimenti conto quanti divisori ci sono tra 2 e $n - 1$; se il risultato è 0 allora il numero è primo
- ▶ **Tattica**:
 - ▶ Uso il pattern del contatore
 - ▶ Per verificare se un numero è un divisore di un altro uso l'operatore %



Soluzione basata sul contatore

```
int numero;
int divisore;
int trovati;

int main() {
    scanf("%d", &numero);

    divisore = 2;
    trovati = 0;

    while (divisore < numero) {
        if (numero % divisore == 0) {
            trovati = trovati + 1;
        }
        divisore = divisore + 1;
    }

    if (trovati > 0 || numero <= 1) {
        printf("Il numero inserito non e' primo");
    }
    else { /* (trovati==0 && numero > 1) */
        printf("Il numero inserito e' primo");
    }
    return 0;
}
```



Decidere se un numero è primo: cercare un divisore

- ▶ In realtà mi basta trovare un divisore tra 2 e $n - 1$
- ▶ Altra idea per il test di primalità di un numero n (**strategia**):
 - ▶ Se il numero è ≤ 1 allora non è primo
 - ▶ Altrimenti verifico che tutti i numeri compresi tra 2 e $n - 1$ non siano divisori di n
- ▶ **Tattica**:
 - ▶ Sto verificando una proprietà universale sul sottoinsieme dei naturali $[2, n - 1]$
 - ▶ Uso il pattern della condizione universale
 - ▶ Quindi assumo che la proprietà sia vera e cerco il controesempio
 - ▶ Assumo che il numero sia primo e cerco un divisore



Soluzione basata sulla verifica di prop. universale

```
int numero;
int divisore;
int primo;

int main() {
    scanf("%d", &numero);
    divisore = 2;
    primo = 1;
    while (divisore < numero) {
        if (numero % divisore == 0) {
            primo = 0;
        }
        divisore = divisore + 1;
    }

    if (primo && numero > 1) {
        printf("Il numero inserito e' primo");
    }
    else { /* (primo==0 || numero <= 1) */
        printf("Il numero inserito non e' primo");
    }
    reutrn 0;
}
```

← Assunzione che non ci siano divisori tra 2 e n-1



Section 4

Altri cicli



Altri cicli

- ▶ Il linguaggio C mette a disposizione altre due strutture di controllo cicliche oltre al **while**
 - ▶ il ciclo `do .. while`
 - ▶ il ciclo `for`
- ▶ I cicli hanno un uso preferenziale, ma se un programma è scritto usando cicli `for` o `do .. while` allora può essere riscritto usando solo il ciclo `while`
 - ▶ vedi slide successive



Il ciclo `do .. while`

- ▶ Sintassi:

do

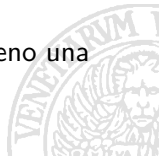
 blocco ;

while (exp) ;

- ▶ Semantica:

1. Esegue il codice di blocco
2. Valuta l'espressione exp. Se è true si ritorna al punto 1.
 altrimenti si termina il ciclo

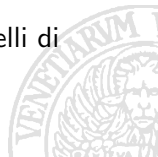
- ▶ A differenza del `while` questo ciclo esegue sempre almeno una volta le istruzioni di blocco



- ▶ Esempio: ripetere la lettura di un valore intero da standard input fino a quando se ne acquisisce uno positivo

```
int a;  
int main() {  
    do {  
        scanf("%d", &a);  
    } while (a <= 0);  
    return 0;  
}
```

- ▶ Attenzione: la condizioni dei cicli in C sono sempre quelli di **permanenza**



Confronto tra while e do .. while

do .. while

```
blocco0;  
do {  
    blocco1;  
} while (exp);  
blocco2;
```

while

```
blocco0;  
blocco1;  
while (exp) {  
    blocco1;  
}  
blocco2;
```



Esempio rivisitato

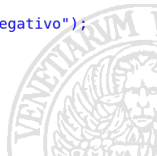
```
int a;
int flag;

int main() {
    flag = 1;
    scanf("%d", &a);
    while (a != 0) {
        if (a < 0) {
            flag = 0;
        }
        scanf("%d", &a);
    }
    if (flag) {
        printf("Tutti positivi");
    }
    else {
        printf("Almeno un numero e' negativo");
    }
}
```

```
int a;
int flag;

int main() {
    flag = 1;
    do {
        scanf("%d",&a);
        if (a < 0) {
            flag = 0;
        }
    } while (a != 0);

    if (flag) {
        printf("Tutti positivi");
    }
    else {
        printf("Almeno un numero e' negativo");
    }
}
```



Ciclo for

► Sintassi:

```
for(inizializzazione; condizione; incremento)  
    blocco;
```

► Semantica:

1. Valuta l'inizializzazione. Di solito sono degli assegnamenti che stabiliscono lo stato delle variabili all'inizio del ciclo. Questa parte può essere assente
 2. Si valuta la condizione. Se risulta false il ciclo termina immediatamente altrimenti si eseguono le istruzioni di blocco
 3. Al termine dell'esecuzione di blocco si valuta l'incremento. Questa parte di solito indica come vanno aggiornate le variabili che determinano il numero di iterazioni del ciclo
- Buon uso del ciclo `for`: dalla lettura dell'intestazione del ciclo si deve sapere quante iterazioni saranno fatte

- ▶ Esempio: leggere 10 numeri da standard input e stamparne la somma in standard output
 - ▶ Conosco il numero di iterazioni!!

```
int i;  
int a;  
int somma;  
int main() {  
    somma = 0;  
    for (i = 0; i < 10; i = i+1) {  
        scanf("%d", &a);  
        somma = somma + a;  
    }  
    printf("La somma e %d \n", somma);  
    return 0;  
}
```



Confronto tra while e for

for

```
blocco0;  
for (inizializza; condizione; aggiorna) {  
    blocco1;  
}  
blocco2;
```

while

```
blocco0;  
inizializza;  
while (condizione) {  
    blocco1;  
    aggiorna;  
}  
blocco2;
```



Section 5

Programmazione strutturata



goto

- ▶ Il linguaggio C mette a disposizione anche la possibilità di **salto incondizionato**
- ▶ L'idea è quella di etichettare delle istruzioni:
 - ▶ `identificatore: istruzione;`
- ▶ Quindi da qualsiasi parte del programma è possibile raggiungere un'istruzione etichettata con il **goto**:
 - ▶ `goto identificatore;`
- ▶ L'uso del goto è deprecato!!

Dijkstra, 1968

The unbridled use of the go to statement has as an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. ... The go to statement as it stands is just too primitive, it is too much an invitation to make a mess of one's program.



Teorema di Böhm Jacopini

- ▶ Il teorema di Böhm Jacopini ha un'importanza fondamentale nella programmazione
- ▶ Formulato nel 1966 ha influenzato la definizione dei linguaggi di programmazione e lo *stile* di programmazione
- ▶ Informalmente il teorema che un qualsiasi algoritmo può essere definito usando solo tre costrutti:
 - ▶ Sequenza
 - ▶ Selezione
 - ▶ Iterazione



Programmazione strutturata

- ▶ La programmazione strutturata emerge alla fine degli anni 60
- ▶ È alla base di molti moderni paradigmi di programmazione (inclusa la programmazione ad oggetti)
- ▶ Rifiuta l'uso del goto e si affida alle strutture di controllo: sequenza, if, while, do while, for
- ▶ Conseguenza del teorema di Böhm Jacopini: **qualsiasi programma scritto usando il goto può essere riscritto senza, a patto di avere a disposizione altri tre tipi di strutture di controllo: sequenza, ripetizione e alternativa**



Ma talvolta i goto sono nascosti...

- ▶ L'istruzione **break**
 - ▶ quando viene eseguita l'esecutore esce dal ciclo più interno
 - ▶ L'istruzione successivamente eseguita è quella immediatamente dopo il blocco del ciclo
- ▶ L'istruzione **continue**
 - ▶ quando viene eseguita l'esecutore salta al test del ciclo senza completare l'esecuzione delle istruzioni del blocco
- ▶ Sono istruzioni *equivalenti* al goto, quindi da evitare



Esempio: test di primalità ed uscita prematura

```
int numero;  
int divisore;  
int primo;  
  
int main() {  
    scanf("%d", &numero);  
    divisore = 2;  
    primo = 1; ← Assunzione che non ci siano  
divisori tra 2 e n-1  
  
    while (divisore < numero) {  
        if (numero % divisore == 0) {  
            primo = 0;  
            break; ← Non serve proseguire la ricerca  
        }  
        divisore = divisore + 1;  
    }  
  
    if (primo && numero > 1) {  
        printf("Il numero inserito e' primo");  
    }  
    else { /* (primo==0 || numero <= 1) */  
        printf("Il numero inserito non e' primo");  
    }  
    return 0;  
}
```

► Riscrivire l'esempio senza la variabile primo



Corretta implementazione in programmazione strutturata

```
int numero;
int divisore;
int primo;

int main() {
    scanf("%d", &numero);
    divisore = 2;
    primo = 1;
    while (divisore < numero && primo) {
        if (numero % divisore == 0) {
            primo = 0;
        }
        divisore = divisore + 1;
    }

    if (primo && numero > 1) {
        printf("Il numero inserito e' primo");
    }
    else { /* (primo==0 || numero <= 1) */
        printf("Il numero inserito non e' primo");
    }
    reutrnr 0;
}
```

← Assunzione che non ci siano divisori tra 2 e n-1

