

3 Costrutti while, for e switch

Questa dispensa propone esercizi sulla scrittura di algoritmi, in linguaggio C, utili alla comprensione dei costrutti `while` e `for` e `switch`.

I costrutti per costruire cicli in C sono il `while` (e la variante `do...while` e il `for`.

```
inizializzazione; //opzionale

while (condizione) {
    corpo;

    incremento; //opzionale
}
```

La condizione è valutata prima di ogni iterazione, inclusa la prima; quindi il corpo del ciclo (e quindi anche l'eventuale istruzione di incremento) potrebbero non eseguire mai, nel caso in cui la condizione sia falsa dall'inizio.

```
inizializzazione; //opzionale

do {
    corpo;

    incremento; //opzionale
} while (condizione);
```

Invece, nella variante `do...while`, il corpo e l'eventuale istruzione di incremento sono eseguiti almeno una volta prima di valutare la condizione.

Il `for` è equivalente, ma ha una sintassi più compatta:

```
for (inizializzazione; condizione; incremento) {
    corpo;
}
```

Si noti che le parentesi sono necessarie solo nel caso di corpo con più di un'istruzione. Inoltre, l'espressione di inizializzazione e di incremento sono opzionali. Di fatto, un ciclo `while` può essere scritto in modo equivalente con un costrutto `for` nel seguente modo:

```
inizializzazione;

for (; condizione; ) {
    corpo;

    incremento;
}
```

Il costrutto del C che permette di scegliere tra alternative multiple (più di 2) è lo `switch`.

```
inizializzazione;

switch (variabile) {
    case valore1:
        istruzione1;
        break;
    case valore2:
        istruzione2;
        break;
    default:
        istruzione3;
        break;
}
```

Il `break` tra un `case` e l'altro permette che venga eseguito solo il codice relativo al `case` in cui si è capitati. Il `break` finale non è necessario ma consigliato (nel caso in cui si dovesse immettere nuovi `case`).

3.0.1 Esercizi

Esercizio 3.1

Scrivere un programma che dato un numero positivo ne restituisca la radice intera

Esercizio 3.2

1. Scrivere un programma che dato un numero reale positivo l e un intero positivo n restituisca l'area del poligono regolare con n lati di lunghezza l . Si implementi una soluzione per che gestisca i casi $n = 3, 4, 5, 6$ e che sia espandibile agevolmente. Suggerimento: l'area del pentagono regolare é:

$$A = l^2 \frac{5}{2} \sqrt{\frac{\sqrt{5}}{10} + \frac{1}{4}}$$

2. Supporre ora che l'utente possa inserire valori negativi per l e n . Si ripeta l'acquisizione dei dati finché l'utente non inserisce dei valori accettabili per i due parametri.

Esercizio 3.3

Scrivere un programma che richiede all'utente un intero positivo e ne stampa a schermo tutti i divisori.

Esercizio 3.4

Scrivere un programma che richiede all'utente un intero positivo e determina se è primo o meno. Il programma deve continuare a chiedere il numero fino a che l'utente non ne inserisce uno positivo.

Esercizio 3.5

Scrivere un programma che richiede all'utente un intero positivo N e stampa a schermo i primi N numeri primi. Ad esempio: con $N = 7$ a schermo avremo 2 3 5 7 11 13 17.

Esercizio 3.6

Scrivere un programma che richiede all'utente un indovinare un numero casuale (generato con la funzione `rand`) tra 1 e 10.

Esercizio 3.7

Dati due numeri m e n questi sono numeri amicali se la somma dei divisori di m è uguale a n , e viceversa (per esempio 220 e 284, 1184 e 1210, 2620 e 2924, 5020 e 5564, 6232 e 6368, 17296 e 18416). Scrivere un programma che ricevuto in ingresso due numeri interi restituisce 1 se i numeri sono amicali, 0 altrimenti.

Soluzioni

Soluzione dell'esercizio 3.1

Abbiamo già visto in un'esercitazione precedente lo pseudocodice simita per questo algoritmo.

Iniziamo con la soluzione utilizzando il costrutto `while`

```
#include <stdio.h>

void main() {

    int n; //numero inserito dall'utente
    int radice; //radice intera di n

    printf("Inserire un numero intero positivo: ");
    scanf("%d",&n);

    radice = n / 2;

    while (radice * radice > n)
        radice--;

    printf("La radice intera di %d e': %d",n,radice);

}
```

Alternativamente si può utilizzare anche il ciclo `for`

```
#include <stdio.h>

void main() {

    int n; //numero inserito dall'utente
    int radice; //radice intera di n
    printf("Inserire un numero intero positivo: ");
    scanf("%d",&n);

    for(radice = n/2; radice*radice > n; radice--) {

    }

    printf("La radice intera di %d e': %d",n,radice);

}
```

Attenzione In questa soluzione il ciclo non è vuoto, ma semplicemente l'unica operazione è il decremento della variabile che viene eseguito nella terza espressione del ciclo for

Soluzione dell'esercizio 3.2

```
1. #include <stdio.h>
#include <math.h>

void main(){

float l; //lato del poligono
int n; //numero di lati
float area; //area del poligono regolare con n lati lunghi l

printf("Inserire la lunghezza del lato ");
scanf("%f",&l);
printf("Inserire il numero di lati ");
scanf("%d",&n);

switch (n) {
case 1:
case 2:
    printf("Il poligono e' degenere");
    break;
case 3:
    area = l * l * sqrt(3) / 2;
    break;
case 4:
    area = l * l;
    break;
case 5:
    area = l * l * 5 / 2 * sqrt(sqrt(5)/10 + 0.25);
    break;
case 6:
    area = l * l * sqrt(3) * 3;
    break;
default:
    printf("Poligono non supportato");
}

if (n > 2 && n < 7)
printf("L'area del poligono regolare con %d lati e' : %f",n,area)
;

}
```

```
2. #include <stdio.h>
#include <math.h>
```

```
void main(){

float l; //lato del poligono
int n; //numero di lati
float area; //area del poligono regolare con n lati lunghi l

do {
    printf("Inserire la lunghezza del lato ");
    scanf("%f",&l);
    printf("Inserire il numero di lati ");
    scanf("%d",&n);
    if (l <= 0 || n < 1)
        printf("Immissione scorretta. Inserire due numeri
                positivi\n");
} while (l <= 0 || n < 1);

switch (n) {
case 1:
case 2:
    printf("Il poligono e' degenero");
    break;
case 3:
    area = l * l * sqrt(3) / 2;
    break;
case 4:
    area = l * l;
    break;
case 5:
    area = l * l * 5 / 2 * sqrt(sqrt(5)/10 + 0.25);
    break;
case 6:
    area = l * l * sqrt(3) * 3;
    break;
default:
    printf("Poligono non supportato");
}

if (n > 2 && n < 7)
printf("L'area del poligono regolare con %d lati e' : %f",n,area)
;

}
```

Soluzione dell'esercizio 3.3

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

```
void main() {
    int n, i;

    do {
        printf("Inserire un intero ");
        scanf("%d" , &n);
    } while(n < 1);

    // soluzione con il ciclo while

    i = 1;

    while(i <= n / 2) {
        if(n % i == 0)
            printf(" %d " , i);

        i++;
    }
}
```

```
#include <stdio.h>

void main() {
    int n, i;

    do {
        printf("Inserire un intero ");
        scanf("%d" , &n);
    } while(n < 1);

    for(i = 1; i <= n / 2; i++)
        if(n % i == 0)
            printf(" %d ", i);
}
```

Attenzione L'init_expr `i = 1;` e la loop_expr `i++;` fanno parte della prima riga nel ciclo. Nel ciclo `while` erano prima del ciclo e all'interno rispettivamente.

Soluzione dell'esercizio 3.4

```
#include <stdio.h>

void main()
{
    int n, i, primo;
```

```

do {
    printf("Inserire un intero: ");
    scanf("%d" , &n);
} while(n < 1);

/*
Variabile di flag: il suo valore cambia da 1 a 0 quando trovo
un
divisore. Nell'inizializzazione della variabile, assumiamo che
il numero inserito sia primo.
*/
primo = 1;

/*
Occorre escludere 1 e n perche' dobbiamo considerare solo i
divisori propri. Quindi iniziamo da 2.
*/
i = 2;

/*
Inutile controllare i numeri da n/2 + 1 a n: non contengono
divisori.

Arresto il ciclo quando la variabile di flag primo diventa 0: è
inutile cercare altri divisori
*/
while (i <= n/2 && primo == 1) {
    if (n % i == 0) //se divisore trovato -> numero non primo!
        primo = 0;

    i++;
}

/*
Qui, al termine del ciclo, se primo == 1 vuol dire che non si è
mai verificato n % i == 0, quindi non esistono divisori propri
ed n è primo.
*/

printf("\n%d", n);

if (primo == 0)
    printf(" non "); // il corpo dell'if è solo quest'istruzione
printf(" è primo\n");
}

```

```

#include <stdio.h>

```

```

void main() {

```



```

int n, i, trovato;

do {
    printf("Inserire un intero: ");
    scanf("%d" , &n);
} while(n < 1);

for (i = n/2; i > 1 && n % i != 0; i--);

/* //In alternativa

    i = n/2;

    while (i > 1 && n % i != 0)
        i--;

*/

if (i > 1)
    printf("Divisore %d trovato: numero non primo.", i);
else //i == 1
    printf("Divisore non trovato: numero
        primo.");

getchar();
}

```

Soluzione dell'esercizio 3.5

```

#include <stdio.h>

void main(){

int n_primi = -2;
int n = 2;
int i;
int divisore;
int flag = 1;

while (n_primi < 1) {
    printf("Numero positivo: ");
    scanf("%d",&n_primi);
}

i = 1;
while (i <= n_primi) {
    flag = 1;
    for(divisore = 2; divisore <= n / 2 && flag == 1; divisore++) {
        if (n % divisore == 0) {

```

```
        flag = 0;
    }
}

    if (flag == 1) {
        printf("%d, ",n);
        i++;
    }
    n++;
}
}
```

Soluzione dell'esercizio 3.6

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

int main ()
{
    int segreto;
    int tentativo;

    srand(time(NULL));

    segreto = rand() % 10 + 1;

    do {
        printf ("Indovina il numero (tra 1 e 10): ");
        scanf ("%d",&tentativo);
        if (segreto < tentativo)
            printf("In numero e' piu' basso\n");
        else {
            if (segreto > tentativo)
                printf("In numero e' piu' alto\n");
        }
    } while (segreto!=tentativo);

    printf("Esatto!");
}
```

Soluzione dell'esercizio 3.7

```
#include <stdio.h>

void main(){
```

```
int n,m;
int sum_n, sum_m;
int i;
scanf("%d",&n);
scanf("%d",&m);

printf("Divisori di %d: ",n);
sum_n = 0;
for (i = 1; i <= n / 2; i++) {
    if (n % i == 0) {
        printf("%d,",i);
        sum_n += i;
    }
}
printf("\n");

printf("Divisori di %d: ",m);
sum_m = 0;
for (i = 1; i <= m / 2; i++) {
    if (m % i == 0) {
        printf("%d,",i);
        sum_m += i;
    }
}
printf("\n");

if (sum_n == m && sum_m == n)
    printf("I due numeri sono amicali\n");
else
    printf("I due numeri non sono amicali\n");
}
```

Attenzione L'algoritmo che viene presentato non è efficiente. Volendo si potrebbe minimizzare il numero delle iterazioni considerate.