

Cicli for

Comandi break, continue, goto

Il costrutto do-while

Il costrutto switch-case

Liceo G.B. Brocchi - Bassano del Grappa (VI)
Liceo Scientifico - opzione scienze applicate
Giovanni Mazzocchin

Il costrutto for

```
int i = 1;
int square = 0;
while (i <= 10) {
    square = i * i;
    cout << square;
    if (i == 10) {
        cout << '\n';
    }
    else {
        cout << '\t';
    }

    i++;
}
```

inizializzazione della variabile di controllo

test di permanenza

riassegnazione della variabile di controllo

possiamo scrivere la stessa cosa in modo più compatto?

Il costrutto for

```
int square = 0;  
for (int i = 1; i <= 10; i++){  
    square = i * i;  
    cout << square;  
    if (i == 10){  
        cout << '\n';  
    }  
    else {  
        cout << '\t';  
    }  
}
```

inizializzazione della variabile di controllo

test di permanenza

riassegnazione della variabile di controllo

quando avviene l'incremento di i?

Il costrutto for

```
int i = 1;
int square = 0;
for (; i <= 10;) {
    int square = i * i;
    cout << square;
    if (i == 10) {
        cout << '\n';
    }
    else {
        cout << '\t';
    }

    i += 1;
}
```

**sintatticamente è corretto... ma
non stiamo sfruttando la
compattezza del costrutto for!**

Il costrutto do-while

```
int decimal_number;  
cin >> decimal_number;  
int binary_digit;
```

```
while (decimal_number != 0) {  
    binary_digit = decimal_number % 2;  
    cout << remainder;  
    decimal_number /= 2;  
}
```

**non viene eseguita alcuna iterazione se
decimal_number == 0.
Ma l'utente potrebbe voler convertire
anche 0!**

Il costrutto do-while

```
int decimal_number;  
cin >> decimal_number;  
int binary_digit;
```

```
if (decimal_number == 0) {  
    cout << 0 << endl;  
}
```

```
while (decimal_number != 0) {  
    binary_digit = decimal_number % 2;  
    cout << binary_digit;  
    decimal_number /= 2;  
}
```

aggiungiamo un controllo per il caso
limite decimal_number == 0

Il costrutto do-while

```
int decimal_number;  
cin >> decimal_number;  
int binary_digit;
```

```
do {  
    binary_digit = decimal_number % 2;  
    cout << binary_digit;  
    decimal_number /= 2;  
} while (decimal_number != 0);
```

il corpo del ciclo viene eseguito almeno una volta perché il controllo di permanenza è posto dopo il corpo! Quindi la conversione sarà eseguita anche con `decimal_number == 0`

corpo del ciclo

test di permanenza

Il costrutto switch-case

```
char grade_c;  
cin >> grade_c;  
int grade_i;  
switch (grade_c) {  
    case 'A':  
        grade_i = 10;  
        break;  
    case 'B':  
        grade_i = 8;  
        break;  
    case 'C':  
        grade_i = 7;  
        break;  
    case 'D':  
        grade_i = 6;  
        break;  
    default: ←  
        grade_i = -1;  
        break;  
}  
cout << "numeric grade is" << grade_i << endl;
```


i vari casi sono identificati da costanti intere (qui sono char, che comunque sono interpretabili come interi)

viene eseguito solo se non si verifica nessuno dei casi identificati dai **case**

Il costrutto switch-case

```
char grade_c;  
cin >> grade_c;  
int grade_i;  
char x = 'A';  
switch (grade_c) {  
    case x:←  
        grade_i = 10;  
        break;  
    case 'B':  
        grade_i = 8;  
        break;  
    case 'C':  
        grade_i = 7;  
        break;  
    case 'D':  
        grade_i = 6;  
        break;  
    default:  
        grade_i = -1;  
        break;  
}
```

x è una variabile, non una costante
il compilatore dà errore

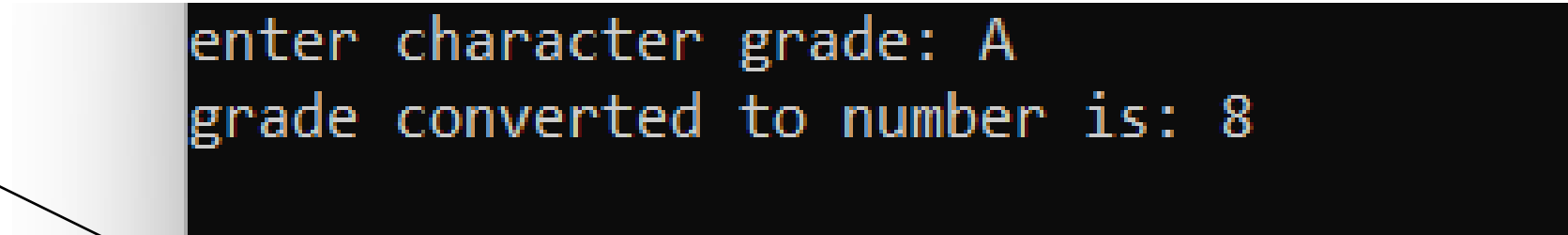


```
ecture2.cpp  
ecture2.cpp(19): error C2051: espressione case non costante
```

Il costrutto switch-case

```
char grade_c;  
cin >> grade_c;  
int grade_i;  
char x = 'A';  
switch (grade_c) {  
    case 'A':  
        grade_i = 10;  
    case 'B':  
        grade_i = 8;  
        break;  
    case 'C':  
        grade_i = 7;  
        break;  
    case 'D':  
        grade_i = 6;  
        break;  
    default:  
        grade_i = -1;  
        break;  
}  
cout << "grade converted to number is: " << grade_i << endl;
```

manca il comando **break**
Ecco cosa succede:



```
enter character grade: A  
grade converted to number is: 8
```

il controllo è passato al case successivo,
che è **case 'B'**

Il costrutto switch-case

```
char grade_c;  
cout << "enter character grade: ";  
cin >> grade_c;  
int grade_i;  
if (grade_c == 'A'){  
    grade_i = 10;  
}  
else if (grade_c == 'B'){  
    grade_i = 8;  
}  
else if (grade_c == 'C'){  
    grade_i = 7;  
}  
else if (grade_c == 'D'){  
    grade_i = 6;  
}  
else {  
    grade_i = -1;  
}  
cout << grade_i;
```

L'istruzione break

```
int in = 0;

while (in != 5) {
    cin >> in;
    cout << in << endl;
    cout << "hello world" << endl;
}
```

la condizione di permanenza è in un unico punto ben definito. Non dobbiamo cercare da nessun'altra parte per capire quando il controllo passerà all'istruzione successiva al ciclo

L'istruzione break

```
int in = 0;
while (true) {
    cin >> in;
    if (in == 5) {
        break;
    }

    cout << in << endl;
    cout << "hello world" << endl;
}
```

La condizione di permanenza indica che il ciclo è infinito. Ma nel corpo effettivamente si esce con il comando **break**.

Poco leggibile e causa di confusione.

Contro i principi della programmazione strutturata

L'istruzione continue

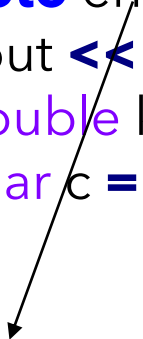
```
for (int i = 1; i <= 10; i++) {  
    if (i == 5) {  
        continue;  
    }  
    cout << i << " ";  
}
```

commenti sull'output?

```
cout << "Output: ";  
1 2 3 4 6 7 8 9 10
```

L'istruzione goto

```
int i = 0;  
int j = 0;  
goto end;  
cout << "hello world";  
double k = 1.0;  
char c = 'c';  
  
end:  
    cout << "jumped to label \"end\"";
```

A black arrow originates from the 'goto end;' line and points to the 'end:' label, illustrating the jump mechanism of the goto statement.

L'istruzione goto

```
int counter = 1;
start_loop:
if (counter > 10){
    cout << '\n';
    goto end_loop;
}
else {
    cout << counter << '\t';
    counter++;
    goto start_loop;
}

end_loop:
cout << "out of goto-like loop";
```

**condizione di uscita dal
«ciclo»**

**jump al controllo di
permanenza nel «ciclo»**

L'istruzione goto

```
int counter = 1;
start_loop:
if (counter > 10) {
    cout << '\n';
    goto end_loop;
}
else {
    cout << counter << '\t';
    counter++;
    goto start_loop;
}

end_loop:
;
```

utilizzare goto porta alla scrittura di codice a «basso livello», poco leggibile e poco manutenibile (ma magari efficiente)
(<https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf>)

Raccolta di esempi ed esercizi

- <https://github.com/Cyofanni/high-school-cs-class/tree/main/C/starter>