- Each case can include more than one statements (no braces are needed !!)

```
switch (grade)
{
 case 'A':
 case 'B': cout << "Good work !!" << endl;
 case 'C':
 case 'D': cout << "Poor work !!" << endl;
          ++numberInTrouble;
 default: cout << grade << " is not a legal letter" << endl;
}
cout << "End of grade assignment" << endl;
```

- To redirect the flow of control outside the *switch* element, a *break* statement must be used

```
switch (grade)
{
 case 'A': cout << "Very good work !!" << endl;
          break;
 case 'B': cout << "Good work !!" << endl;
          break;
 case 'C': cout << "Average work !!" << endl;
          break;
 case 'D': cout << "Poor work !!" << endl;
          break;
 case 'F': cout << "Very poor work !!" << endl;
          break;
 default: cout << grade << " is not a legal letter" << endl;
}
cout << "End of grade assignment" << endl;
```

- *switch* vs *if* statements

```
if(grade == 'A')
  cout << "Very good work !!" << endl;
else if(grade == 'B')
  cout << "Good work !!" << endl;
else if(grade == 'C')
  cout << "Average work !!" << endl;
else if(grade == 'D')
  cout << "Poor work !!" << endl;
else if(grade == 'F')
  cout << "Very poor work !!" << endl;
else cout << grade << " is not a legal letter" << endl;
cout << "End of grade assignment" << endl;
```

- The *do-while* statement

```
do (Expression)
  Statement
while (Expression);
```

- Statement is executed first without testing any condition

- If the value of the -logical- expression is *true*, Statement is executed again

- If the value of the expression is *false*, execution continues at the statement immediately following the loop

- Compound *do-while* statement

```
do
{
  Statement1;
  Statement2;
 ...
  StatementN;
} while (Expression);
```

- *while* statement vs *do-while statement*

- In the case of the *while* statement, the condition is tested in the beginning

- In the case of the *do-while* statement, the condition is tested at the end

- As a result, the body of the *do-while* statement will be executed at least one time !!!

- The *for* statement

  - A looping control structure

  - Simplifies the design of count-controlled loops

  - Examples

```
for(count = 1; count <= 10; count++)
  cout << count << endl;

for(count = 10; count <= 1; count--)
  cout << count << endl;
```

- Any *for* statement corresponds to a *while* statement

- Compound *for* statement

```
for(InitStatement; LogicalExpression1; Expression2) br {
  Statement1;
  Statement2;
 ...
  StatementN;
}
```

- Nested *for* statement

- The *break* statement

  - It is used with the *switch* statement and with loop statements (i.e., while, for, do-while)

  - It causes an immediate exit from the *switch* or from the loop

```
sum=0;
for (i=0; i<=10; i++)
{
   cout << "Enter a number";
   cin >> num;
   if(num == -9999)
     break;
   sum = sum + num
}
```

- What if we have a nested loop statement ?

- *break* causes an immediate exit from the innermost loop only !!

```
sum=0;
for (i=0; i<=10; i++)
{
   cout << "Enter a number";
   cin >> num;
   for(j=0; j<=10; j++)
   {
     if(sum  > 1000)
       break;
     sum =  sum * j * num;
    }
}
```

- The *continue* statement

  - It is used with loop statements (i.e., while, for, do-while)

  - It causes an immediate branch to the bottom of the loop

```
sum=0;
for (i=0; i<=10; i++)
{
   cout << "Enter a number";
   cin >> num;
   if(num == -9999)
     continue;
   sum = sum + num
}
```

Recommended exercises

- *Quick Check* excersices: 1, 2, 3, 5

- *Exam preparation* excersices: 5, 6, 7, 9, 10, 11, 12