==================================================================

**CONTROL STRUCTURES (cont.)**

==================================================================

## Repetition

- When an action (or a sequence of actions) must be repeated, a **loop** is used
- A **loop** is a program construction that
  repeats a statement or sequence of statements a number of times
- C++ includes several ways to create loops
  - **while** loops
    - the statement(s) in the loop are executed zero or more times
  - **do … while** loops
    - the statement(s) in the loop are executed at least once
  - **for** loops
    - the statement(s) in the loop are executed zero or more times
- **while** loops and **do … while** loops are
  typically used for sentinel-controlled repetition
- **for** loops are
  typically used for counter-controlled repetition

### • while statement

```
while (boolean_expression)
   statement
```

- Example 1 (sum list of positive values)

```
int value, sum = 0;
cout << "Value? ";
cin >> value;
while (value > 0)     // when does it end?
{
  sum = sum + value;  // OR sum += value;
  cout << "Value? ";
  cin >> value;
}
cout << "Sum = " << sum << endl;
```

- **do ... while statement**

```
do
   statement
while (boolean_expression) ;     ← NOTE the semicolon
```

  - Example (printing ASCII CODES)

```
char symbol;
const char STOP = '#'; // NOTE: good programming practice

do
{
  cout << "Letter/digit (" << STOP << " to QUIT) ? ";
  cin >> symbol;
  cout << "ASCII(" << symbol << ") = " << (int)symbol << endl;
} while (symbol != STOP);
```

    - Question: how to prevent ASCII('#') from being shown?

- **for statement**

```
for (initializing_expr; boolean_expr; step_exp)
   statement
```

  - Example 1 (sum of all integer values in the range [1..100])

```
int i;
int sum = 0;   // DON'T FORGET INITIALIZATION !!!

for (i=1; i<=100; i=i+1)
  sum = sum + i;

cout << "1 + 2 + ... + 100 = " << sum << endl;
```

  - Example 2 (sum of any 5 integer values, read from the keyboard)

```
int sum = 0;

for (int i=1; i<=5; i++)  // i is only visible inside the block
{
  int value;              // value only visible inside the block
  cout << "Value no. " << i << " ? ";
  cin >> value;
  sum = sum + value;
}

cout << "Sum of entered values = " << sum << endl;
```

  - NOTE: any of the expressions in a **for** statement can be ommited.

- `break and continue statements`
  - `break`
    - leaves a loop, even if the condition for its end is not fulfilled
    - It can be used to end an infinite loop
    - can be used with any type of loop

```
for (int divisor=2; divisor<=num; divisor++)
{
  if (num % divisor == 0)
  {
    cout << "First integer divisor (<> 1) of " << num <<
            " is " << divisor << endl;
    break;
  }
}
```

  - `continue`
    - causes the program to skip the rest of the loop
      in the current iteration,
      as if the end of the statement block had been reached,
      causing it to jump to the start of the following iteration
    - can be used with any type of loop

```
for (int i=1; i<=100; i++)
{
  if (i==13) continue; //... I'm not superstitious, but...
  cout << i << endl;
}
```

- NOTES:
  - If you have a loop within a loop, and a `break` in the innner loop,
    then the `break` statement will only end the inner loop.
  - The use of `break` and `continue` in lengthy loops,
    makes the code difficult to read ... :-(

- **"infinite" loops**
  - `while (true) {...};`
  - `while (1) {...};`
  - `do { ... } while (true);`
  - `do { ... } while (1);`
  - `for (;;) {...};`
  - In fact, no loop should be infinite,
    otherwise the program would never end ...
  - An "infinite" loop should contain a **break** statement somewhere.
  - Sometimes, there are some infinite loops caused by coding errors ...

=============================================================

## INPUT – dealing with invalid inputs

=============================================================

### Invalid inputs

- Consider the following code:

```
int sum = 0;

for (int i=1; i<=5; i++)  // i is only visible inside the block
{
  int value;              // value only visible inside the block
  cout << "Value no. " << i << " ? ";
  cin >> value;
  sum = sum + value;
}
```

- What happens if a careless user inserts value **1o** instead of **10**?
- … the loop becomes an endless loop!
- In the lectures it will be explained in detail why this happens.
- In summary:
    - the **o** (lowercase letter) is not compatible with an integer value
    - the input will fail and the input stream will enter a failure state;
    - the failure state must be cleared, so that more input can take place
    - even if the failure state is cleared
      the **o** could only be extracted to a **char** or **string** variable
    - so, if one wants to continue reading integer values,
      the **o** must be removed from the input buffer

### Testing for input failure and clearing invalid input state

- The easist way to test whether a stream is okay is to test its "truth value":
    - `if (cin) ... // OK to use cin, it is in a valid state`
    - `if (! cin) ... // cin is in an invalid state`
    - `while (cin >> value) ... // OK, read operation successful`

- Alternative way:

    -
      ```
      cin >> value;
      if (! cin.fail()) ... // OK; input did not fail
      ```
- Other condition states can be tested (*to be presented later*):
    - `cin.good(), cin.bad(), cin.eof()`
- The **clear** operation puts the I/O stream condition back in its valid state
    - `if (cin.fail()) cin.clear(); //NOTE: it does not clean the buffer`

## Cleaning the input buffer

- Sometimes, it is necessary to remove from the input buffer the input that caused the failure. This is done using the `cin.ignore()` call.
- It can be called in 3 different ways:
  - `cin.ignore()`
    - a single character is taken from the input buffer and discarded
  - `cin.ignore(numChars)`
    - the number of characters specified are taken from the input buffer and discarded;
  - `cin.ignore(numChars, delimiterChar)`
    - discard the number of characters specified, or
      discard characters up to and including the specified delimiter
      (whichever comes first):
    - Example: `cin.ignore(10,'\n');`
      - ignore 10 characters or to a newline, whichever comes first
- The whole buffer contents can be cleaned by calling:
  - `cin.ignore(numeric_limits<streamsize>::max(),'\n');`
    `// => #include <ios>   AND   #include <limits>`
- Note: **BE CAREFUL!!!**
  a call to `cin.ignore()` when the buffer is empty
  will stop the execution of the program until something is entered !!!

## Other input operations

- `cin.get()`
  - returns the next character in the stream
- `cin.peek()`
  - returns the next character in the stream,
    but does not remove it from the stream

### TO DO BY STUDENTS

- Write a program that uses `cin >>` , `cin.peek()` and `cin.ignore()`
  to read the integer number contained in "#ABcdE12345$Esc", that is 12345,
  discarding all the remaining symbols.

```
#include <iostream>

using namespace std;

int main()
{
  int num = 0;
  char ch;
  ch = cin.peek();
  while (ch != '\n')
  {
    cout << ch;
    if (ch >= '0' && ch <= '9')
      num = num * 10 + ch - '0';
    cin.ignore(1);
    ch = cin.peek();
  }

  cout << endl;
  cout << "num =" << num << endl;

  return 0;
}
```

```
/*
 USING REPETITION STATEMENTS: FOR
*/

#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
      const unsigned int NUMBER_PRECISION = 3;
      double operand1, operand2;  // input operands
      char operation; // operation; possible values: + - * /
      double result; // result of "operand1 operation operand2"
      unsigned int i;
      unsigned int numOperations;

      cout << "Number of operations to do ? ";
      cin >> numOperations;

      for (i=1; i<=numOperations; i++)   // OR for (unsigned int i=1; ...)
      {
            // input 2 numbers
            cout << endl;
            cout << "x op y ? ";
            cin >> operand1 >> operation >> operand2;

           bool validOperation = true; // assume operation is valid
            // compute result if operation is valid
            switch (operation)
            {
            case '+':
                  result = operand1 + operand2;
                  break;
            case '-':
                  result = operand1 - operand2;
                  break;
            case '*':
                  result = operand1 * operand2;
                  break;
            case '/':
                  result = operand1 / operand2;
                  break;
            default:
                  validOperation = false;
            }

            //show result or invalid input message
            if (validOperation)
            {
                  cout << fixed << setprecision(NUMBER_PRECISION);
                  cout << operand1 << ' ' << operation << ' ' << operand2 <<
                        " = " << result << endl;
            }
            else
                  cerr << "Invalid operation !\n";
      }
      return 0;
}
===========================================================================
/*
 USING REPETITION STATEMENTS: WHILE
*/
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
      const unsigned int NUMBER_PRECISION = 3;
      double operand1, operand2;  // input operands
      char operation; // operation; possible values: + - * /
      double result; // result of "operand1 operation operand2"
      unsigned int i;
      unsigned int numOperations;

      cout << "Number of operations to do ? ";
      cin >> numOperations;

      i = 0; // counter              //COMMON ERRORS: - forget initialization
      while (i < numOperations)      //               - off by one loops
      {
            // read operation
            cout << endl;
            cout << "x op y ? ";
            cin >> operand1 >> operation >> operand2;

           bool validOperation = true; // assume operation is valid
            // compute result if operation is valid
            switch (operation)
            {
            case '+':
                  result = operand1 + operand2;
                  break;
            case '-':
                  result = operand1 - operand2;
                  break;
            case '*':
                  result = operand1 * operand2;
                  break;
            case '/':
                  result = operand1 / operand2;
                  break;
            default:
                  validOperation = false;
            }

            //show result or invalid input message
            if (validOperation)
            {
                  cout << fixed << setprecision(NUMBER_PRECISION);
                  cout << operand1 << ' ' << operation << ' ' << operand2 <<
                        " = " << result << endl;
            }
            else
                  cerr << "Invalid operation !\n";
            i = i+1; // OR i++;  OR i += 1;  // DO NOT FORGET ... TO UPDATE
      }
      return 0;
}
```

```
/*
 USING REPETITION STATEMENTS: DO ...WHILE
*/
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
      const unsigned int NUMBER_PRECISION = 3;
      double operand1, operand2;  // input operands
      char operation; // operation; possible values: + - * /
      double result; // result of "operand1 operation operand2"
      unsigned int i;
      unsigned int numOperations;

      cout << "Number of operations to do ? ";
      cin >> numOperations;

      i = 0; // counter
      do  // WHAT HAPPENS IF USER INSERTS numOperations = 0 ?!
      {
            // read operation
            cout << endl;
            cout << "x op y ? ";
            cin >> operand1 >> operation >> operand2;  // TRY WITH 'a + 2'
            //cout << "OP = " << operand1 << operation << operand2 << endl;

           bool validOperation = true; // assume operation is valid
            // compute result if operation is valid
            switch (operation)
            {
            case '+':
                  result = operand1 + operand2;
                  break;
            case '-':
                  result = operand1 - operand2;
                  break;
            case '*':
                  result = operand1 * operand2;
                  break;
            case '/':
                  result = operand1 / operand2;
                  break;
            default:
                  validOperation = false;
            }

            //show result or invalid input message
            if (validOperation)
            {
                  cout << fixed << setprecision(NUMBER_PRECISION);
                  cout << operand1 << ' ' << operation << ' ' << operand2 <<
                        " = " << result << endl;
            }
            else
                  cerr << "Invalid operation !\n";
            i = i+1; // OR i++;  OR i += 1;
      } while (i<numOperations);
      return 0;
}
```

```
============================================================================
/*
USING REPETITION STATEMENTS  - user is not asked for the no. of operations
*/
#include <iostream>
#include <iomanip>
#include <cctype>  // for using toupper()

using namespace std;

int main()
{
     const unsigned int NUMBER_PRECISION = 3;
     double operand1, operand2;  // input operands
     char operation; // operation; possible values: + - * /
     double result; // result of "operand1 operation operand2"
     char anotherOperation;

     do
     {
          // read operation
          cout << endl;
          cout << "x op y ? ";
          cin >> operand1 >> operation >> operand2;

          // compute result if operation is valid
         bool validOperation = true; // assume operation is valid
          switch (operation)
          {
          case '+':
               result = operand1 + operand2;
               break;
          case '-':
               result = operand1 - operand2;
               break;
          case '*':
               result = operand1 * operand2;
               break;
          case '/':
               result = operand1 / operand2;
               break;
          default:
               validOperation = false;
          }

          //show result or invalid input message
          if (validOperation)
          {
               cout << fixed << setprecision(NUMBER_PRECISION);
               cout << operand1 << ' ' << operation << ' ' << operand2 <<
                    " = " << result << endl;
          }
          else
               cerr << "Invalid operation !\n";

          cout << "Another operation (Y/N) ? ";
          cin >> anotherOperation;
          anotherOperation = toupper(anotherOperation);

     } while (anotherOperation == 'Y');

     return 0;
}
```

```
============================================================================
/*
USING REPETITION STATEMENTS
- NESTED LOOPS
- DEALING WITH INVALID INPUTS   → Teaching note: START WITH SIMPLE EXAMPLE
*/
#include <iostream>
#include <iomanip>
#include <cctype>  // for using toupper()

using namespace std;

int main()
{
     const unsigned int NUMBER_PRECISION = 3;

     double operand1, operand2;  // input operands
     char operation; // operation; possible values: + - * /
     double result; // result of "operand1 operation operand2"
     bool validOperation; // operation is + - * or /
     char anotherOperation;

     do
     {
          // input 2 numbers and the operation
          bool validOperands = false;  // ONLY VISIBLE INSIDE ABOVE "do" BLOCK
          do
          {
               cout << endl << "x op y ? ";
               if (cin >> operand1 >> operation >> operand2)
                    validOperands = true;
               else
               {
                    cin.clear();           // clear error state
                    cin.ignore(1000,'\n'); // clean input buffer
               }
          } while (!validOperands);


          // compute result if operation is valid

          validOperation = true;

          switch (operation)
          {
          case '+':
               result = operand1 + operand2;
               break;
          case '-':
               result = operand1 - operand2;
               break;
          case '*':
               result = operand1 * operand2;
               break;
          case '/':
               result = operand1 / operand2;
               break;
          default:
               validOperation = false;
          }
```

```
            //show result or invalid operator message
            if (validOperation)
            {
                  cout << fixed << setprecision(NUMBER_PRECISION);
                  cout << operand1 << ' ' << operation << ' ' << operand2 <<
                        " = " << result << endl;
            }
            else
                  cerr << "Invalid operator !\n";

            cout << "Another operation (Y/N) ? ";
            cin >> anotherOperation;
            anotherOperation = toupper(anotherOperation);

      } while (anotherOperation == 'Y');

      return 0;
}
```

==========================================================================

**TO DO BY STUDENTS:**

- modify the "**`do ... while (!validOperands);`**" loop so that the initial value of **`validOperands`** is **`true`**
- modify the "**`do ... while (!validOperands);`**" loop so that it is also repeated when the operator is not valid

ANS: `while (!validNumbers || !(operation=='+' || operation=='-' || operation=='*' || operation=='/' ))`

```
/*
USING REPETITION STATEMENTS
DETECTING END OF INPUT (CTRL-Z)
*/
#include <iostream>
#include <iomanip>
#include <cctype>

using namespace std;

int main()
{
     const unsigned int NUMBER_PRECISION = 6;

     double operand1, operand2;
     char operation;
     double result;
     bool anotherOperation;
    //ANY OF THESE VARIABLES COULD HAVE BEEN DECLARED ELSEWHERE ?

     do
     {
          bool validOperands; // ONLY VISIBLE INSIDE THE 'green' do ...while cycle
          anotherOperation = true;

          do
          {
               cout << endl << "x op y (CTRL-Z to end) ? ";
               cin >> operand1 >> operation >> operand2;
               validOperands = true;
               if (cin.fail())
               {
                    validOperands = false;
                    if (cin.eof())  // use cin.eof() only after cin.fail() returns TRUE
                         anotherOperation = false; //ALTERNATIVE: return 1;
                    else
                    {
                         cin.clear();
                         cin.ignore(1000,'\n');
                    }
               }
               else
                    cin.ignore(1000,'\n'); //clear any additional chars
          } while (anotherOperation && !validOperands);

          //above cycle is equivalent to:
          //REPEAT ... UNTIL ((NOT anotherOperation) OR validOperands)
          //sometimes it is easier to think in terms of REPEAT...UNTIL...
          //then negate REPEAT condition to obtain the WHILE condition


          if (validOperands)
          {
               bool validOperation = true;
               switch (operation)
               {
               case '+':
                    result = operand1 + operand2;
                    break;
               case '-':
```

```
                        result = operand1 - operand2;
                        break;
                case '*':
                        result = operand1 * operand2;
                        break;
                case '/':
                        result = operand1 / operand2;
                        break;
                default:
                        validOperation = false;
                }

                //show result or invalid input message
                if (validOperation)
                {
                        cout << fixed << setprecision(NUMBER_PRECISION);
                        cout << operand1 << ' ' << operation << ' ' <<
operand2 <<
                                " = " << result << endl;
                }
                else
                        cerr << "Invalid operation !\n";

            }

        } while (anotherOperation);   // EQUIVALENT TO: repeat ... until (??);

        return 0;
}
```

NOTE- alternative way to invoque cin.ignore():

**std::cin.ignore ( std::numeric_limits<std::streamsize>::max(), '\n' );**

OR JUST

cin.ignore ( numeric_limits<streamsize>::max(), '\n' );

This requires you to include the following header files:

```
#include <iostream>
#include <ios>        // for streamsize
#include <limits>     // for numeric_limits
```

```
/*
  MORE ON REPETITION (OR ITERATION) STATEMENTS

  CALCULATE THE SQUAREROOT USING A BABILONIAN ALGORITHM
  SEE PROBLEM 2.14
*/
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   cout << "Please enter a number: ";
   double num;
   cin >> num;

   // VARIABLES CAN BE DECLARED ANYWHERE, IN THE SAME CODE BLOCK,
   // BEFORE THEY ARE USED
   const double EPSILON = 1E-6;
   double xnew = num;
   double xold;

   do
   {
      xold = xnew;
      xnew = (xold + num / xold) / 2;
   }
   while (fabs(xnew - xold) > EPSILON);

   cout << "The square root is " << xnew << "\n";
   return 0;
}
```

**num = 2**

| xold | xnew |
|---|---|
| 2 | 1.5 |
| 1.5 | 1.41667 |
| 1.41667 | 1.41422 |
| 1.41422 | 1.41421 |
| 1.41421 | ... |
| ... | ... |

**TO DO BY STUDENTS:**
**- specify a maximum number of iterations** (SEE PROBLEM 2.14)

```
/*
SOME LOOP VARIATIONS

COMMA OPERATOR

BREAK AND CONTINUE STATEMENTS

*/
#include <iostream>

using namespace std;

int main()
{
	int x, y;
	int i, j;

	// FOR - 1
	//------------------------------------------------------------------
	cout << "FOR - 1\n";
	for (int k = 10; k != 0; k = k-2) // WHAT DOES IT DO ?

		cout << "k = " << k << endl;


	//COMMA OPERATOR
	//------------------------------------------------------------------
	cout << endl << "COMMA OPERATOR\n";
	x = (y=3, y+1); // the parentheses are necessary,
	                // because the comma operator as lower precedence
	               // than the assignment operator
	cout << "x = " << x << "; y = " << y << endl;


	// FOR - 2
	//------------------------------------------------------------------
	cout << endl << "FOR - 2\n";
	for (i=1, j=10; i < j; i++, j--) // note the comma operator
		cout << "i = " << i << "; j = " << j << endl;



	// FOR - 3 (infinite...? loop)
	//------------------------------------------------------------------
	cout << endl << "FOR - 3\n";
	for (;;)
	{
		char letter;
		cout << "letter (q-quit) ? ";
		cin >> letter;
		if (letter == 'q' || letter =='Q')
			break;
		//do something
		cout << "WORKING HARD !\n";
		//...
	}
	cout << "You entered 'q' or 'Q' \n";
	// WHILE - 1
	//----------------------------------------------------------------------
	cout << endl << "WHILE - 1\n";
	char letter;
	cout << "letter (q-quit) ? ";
	cin >> letter;
	while (letter != 'q' && letter !='Q')
	{
		//do something
		cout << "WORKING HARD !\n";
		//...
		cout << "letter (q-quit) ? ";
		cin >> letter;
	}
	cout << "You entered 'q' or 'Q' \n";


	// WHILE - 2 (infinite...? loop)
	//----------------------------------------------------------------------
	cout << endl << "WHILE - 2\n";
	while (true) // OR while (1)
	{
		char letter;
		cout << "letter (q-quit) ? ";
		cin >> letter;
		if (letter == 'q' || letter =='Q')
			break;
		//do something
		cout << "WORKING HARD !\n";
		//...
	}
	cout << "You entered 'q' or 'Q' \n";

	// FOR - 4 (a strange FOR loop...!) DON'T DO ANYTHING LIKE THIS
	//----------------------------------------------------------------------
	cout << endl << "FOR - 4\n";
	int n;
	for (cout << "n (0=end)? "; cin >> n, n != 0; cout << "n (0=end)? ")
		cout << n*10 << endl;


	// FOR - 5 - NOT RECOMMENDED
	//----------------------------------------------------------------------
	cout << endl << "FOR - 5\n";
	const unsigned int NUM_VALUES = 10;
	double sum = 0, value, mean;

	i = 1;
	for ( ;i <= NUM_VALUES; ) // NOT RECOMMENDED
	{
		cout << "n" << i << "? ";
		cin >> value;
		sum = sum + value;
		i++;
	}

	mean = sum / NUM_VALUES;
	cout << "mean value = " << mean << endl;

	return 0;
}
```

```
/*
LOOP pitfalls
Be careful !!!

*/
#include <iostream>

using namespace std;

int main()
{
  // Guess what do the loops do:

  // PITFALL 1
  for (int count = 1; count <= 10; count++); // NOTE the semicolon
    cout << "Hello\n";

  // PITFALL 2
  int x = 1;
  while (x != 12)
  {
    cout << x << endl;
    x = x + 2;
  }

  // PITFALL 3
  float y = 0;
  while (y != 12.0)
  {
    cout << y << endl;
    y = y + 0.2;
  }

  return 0;
}
```

==========================================================================

**TO DO BY STUDENTS:**

Write a program to show the multiplication tables, from 2 to 9:

```
2x1 =  2
2x2 =  4
...
2x9 = 18
--------

3x1 =  3
...
--------
9x1 =  9
...
9x8 = 72
9x9 = 81
```