DIGITAL ARTS & ENTERTAINMENT

Overview course Programming 1

1. Variables 1
2. Variables 2
3. Variables 3
4. Conditionals
5. Iterations
6. Functions 1
7. Functions 2
8. Arrays
9. Strings – Game
10. Classes 1 – Encapsulation
11. Classes 2 – Static const

howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

Iterations

howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

Iterations

- while
- do while
- for loop

Iterations

- Purpose: to implement an automatically repeating block of code.
- We won't write:

```
DrawLine(0, 10, 100, 10);
DrawLine(0, 20, 100, 20);
DrawLine(0, 30, 100, 30);
DrawLine(0, 40, 100, 40);
DrawLine(0, 50, 100, 50);
DrawLine(0, 60, 100, 60);
//   . . .
DrawLine(0, 100, 100, 100);
```

Iterations

- We will write:

```
REPEAT_A_NUMBER_OF_TIMES
{
  DrawLine(0, y, 100, y);
  y += 2;
}
```

How?

- Three ways:
  - The while-structure.
  - The do-while structure.
  - The for-structure.

The while-structure is also called the while-loop, or simply the "while". Same for the do-while and the for.

The while-structure

- In principle the easiest of the three.
- General form:

```
while ( BOOLEAN EXPRESSION )
{
   // write any code here: the loop
}
```

```
while (BOOLEAN EXPRESSION )
{
        // write any code here: the loop
}
```

Mechanism:

0) Execute code until we come to a while-structure.

1) The boolean expression is evaluated:

2a) If the result is *true*, then the loop is executed.

2b) If the result is *false*, then the code quits the structure and continues with the rest of the program.

3) If the result is *true*, then the loop is executed (see 2a). After the loop is executed, the code jumps back to (1) and continue from there (the boolean expression is evaluated again, etc.).

```
while (BOOLEAN EXPRESSION )
{
      // write any code here: the loop
}
```

Example:

```
int y { 20 };
while (y < 200 )
{
     DrawLine( 20, 20, 200, y );
     y += 20;
}
```

```
while (BOOLEAN EXPRESSION )
{
        // write any code here: the loop
}
```

Remarks:

- It's possible to end up in an *infinite loop* : if the code enters the loop, if will only exit if the boolean expression returns false at a later point.

```
while (BOOLEAN EXPRESSION )
{
        // write any code here: the loop
}
```

Remarks:

- It's possible to end up in an *infinite loop* : if the code enters the loop, if will only exit if the boolean expression returns false at a later point.
- If the boolean expression is written in such a way that it always returns true, then the program will get stuck in the loop.

```
while (BOOLEAN EXPRESSION )
{
        // write any code here: the loop
}
```

Remarks:

- It's possible to end up in an *infinite loop* : if the code enters the loop, if will only exit if the boolean expression returns false at a later point.
- If the boolean expression is written in such a way that it always returns true, then the program will get stuck in the loop.
- An infinite loop does not happen by accident: you as the programmer have to take care that this doesn't happen!

The do-while structure

- The little brother of the while-structure
- General form:

```
do
{
    // write any code here: the loop
}
while ( BOOLEAN EXPRESSION );
```

semicolon !!

```
do
{
        // write any code here: the loop
}
while (BOOLEAN EXPRESSION );
```

Mechanism:

0) Execute code until we come to a while-structure.

1) The loop is executed.

2) The boolean expression is evaluated:

3a) If the result is *true*, then the code jumps back to (1) and continues from that point.

3b) If the result is *false*, then the code exits the structure and continues with the rest of the program.

```
do
{
      // write any code here: the loop
}
while (BOOLEAN EXPRESSION );
```

Example:

```
int y { 20 };
do
{
     DrawLine( 20, 20, 200, y);
     y += 20;
}
while (y < 200 ) ;
```

```
do
{
        // write any code here: the loop
}
while (BOOLEAN EXPRESSION );
```

Remarks:

- The loop is always executed at least once in the case of a do-while.
- When using a while, it is possible that the loop is never executed.
- Do not forget the semicolon at the end of the do-while. However, don’t write a semicolon at the end of a while-loop (confusing).

The for-structure

- A controlled way of writing an iterative loop.
- Example:

```
for ( int count { 20 }; count < 200; count += 20 )
{
    DrawLine( 20, 20, 200, count );
}
```

```
for ( START ; CHECK ; CONTINUE)
{
        BLOCK
}
```

Mechanism:

0) Execute code until we come to a for-structure.

1) START is executed.

2) CHECK has to be a boolean expression. It is evaluated:

3a) If the result is *true*, then BLOCK is executed.

3b) If the result is *false*, then the code exits the structure and continues with the rest of the program.

4) After the BLOCK is executed, then CONTINUE is executed next. After this the code jumps back to CHECK (2) and continues from there on.

Typical situation

```
int sum { 0 };
for ( int count { 0 }; count < 100 ; ++count)
{
        sum += count;
}
```

each iteration, count is increased by 1

> sum now contains the sum of these numbers … ?

Other situations

```
int sum { 0 };
for ( int count { 100 }; count >= 0 ; --count)
{
        sum += count;
}
```

each iteration, count is decreased by 1

> sum now contains the sum of these numbers … ?

Other situations

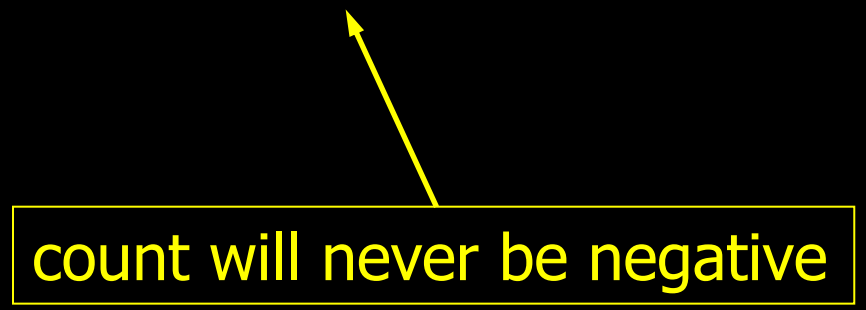
```
for ( float angle{ 0.f }; angle < 6.28f ; angle += 0.01f)
{
	std::cout << angle << "  " << sin(angle) << '\n';
}
```

Problem?

```
int sum { 0 };
for ( unsigned int count { 100 }; count >= 0 ; --count)
{
	sum += count;
}
```

Problem?

```
int sum { 0 };
for ( unsigned int count { 100 }; count >= 0 ; --count)
{
        sum += count;
}
```



Tips:

- Use the do-while loop when the loop must at least be executed once.
- Use do-while and while when different situations can cause the loop to end
- Use the for loop if there is the need to control how many times the loop must be executed.