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

Topic

- Selections
- Enumerations

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- Purpose: to make a program do different things depending on the situation.
- Example:

```
int number{ 6 };
string month;
if (number == 6 ) month = "June";
```

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

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- General: if (condition) statement
  - condition is true → statement executed
  - condition is false → statement ignored
  - condition is a *boolean expression*

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

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- Example:

```
int number { 2 };
if ( number < 5 ) number = 5;
std::cout << number;
```

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

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- Example:

```
int number { 2 };
if ( true ) number = 5;
std::cout << number;
```

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

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- Example:

```
int number { 2 };
if ( number > 15 ) number = 5;
std::cout << number;
```

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

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- Example:

```
int number { 2 };
if ( false ) number = 5;
std::cout << number;
```

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

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- **Compound** statement:

```
if(number < 5)
{
        number = 5;
        result = 15 / 7 * 6 – 8 + 5;
}
```

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

Conditional structure: if and else

- DANGER:

```
number = 10;
if(number < 5)
      number = 5;
      number += 1;
```

- What is the value of number?
- Is this a simple or a compound statement?

Conditional structure: if and else

- DANGER:

```
number = 10;
if(number < 5)
      number = 5; → simple statement !!!
number += 1; → allways executed

→number → 11
```

Conditional structure: if and else

- Suggestion: write simple statement on same line!
  - Prevents bugs
  - Enhances the readability

```
number = 10;
if(number < 5)number = 5;
number+=1;
```

Conditional structure: if and else

- if(condition) statement1 else statement2
- Using simple statements:

```
if ( condition ) number = 1;
else number = 2;
```

Conditional structure: if and else

- if(condition) statement1 else statement2
- Using compound statements:

```
if ( condition )
{
    // execute if condition is true
    number = 1;
}
else
{
    // execute if condition is false
    number = 2;
}
```

DIGITAL ARTS & ENTERTAINMENT

Conditional structure: if and else

- Composite form:

```
if (condition1 )
{
     // execute if condition1 is true
}
else  if (condition2 )
{
     // execute if condition2 is true
}
else  if (condition3 )
{
     // execute if condition3 is true
}
else
{
     // execute if condition3 is= false
}
```

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

The selective structure: switch

- The switch-structure is an alternative way of writing an if-structure.
- Pros: sometimes faster / easier.
- The switch structure compares the contents of an integer typed variable with a list of given values, and executes based on the result.

The switch-structure

```
switch ( int-equivalent variable )
{
        case value1 :
                // statements
                break;
        case value2:
                // statements
                break;
        default:
                // statements
}
```

The variable has to be an *int*, or another type that can be treated as an *int* (eg. char).

The switch-structure

```
switch ( int-equivalent variable )
{
        case value1 :
                // statements
                break;
        case value2:
                // statements
                break;
        default:
                // statements
}
```

The *break;* statements are optional.
If a case block isn't terminated by a *break*, the code will keep on executing lines from the next case, until it hits a *break*, or the end of the switch.

The switch-structure

```
switch ( int-equivalent variable )
{
        case value1 :
                // statements
                break;
        case value2:
                // statements
                break;
        default:
                // statements
}
```

The *default:* part is optional. This is where you write the code that needs to be executed if none of the given values match.

The switch-structure

Example

```
switch(x){
  case 91:
  case 12:
  case 3:
    g_Text = "the value of x is 91 or 12 or 3";
    break;
  default:
    g_Text = "the value of x is not 91 not 12 not 3";

}
```

The switch-structure

switch example	if-else equivalent

`switch (x) {`
`  case 1:`
`  g_Text = "x is 1";`
`  break;`
`  case 2:`
`  g_Text = "x is 2";`
`  break;`
`  default:`
`  g_Text = "x is unknown";`
`}`
	`if(x == 1)`
	`{`
	`  g_Text = "x is 1";`
	`}`
	`else if(x == 2)`
	`{`
	`  g_Text = "x is 2";`
	`}`
	`else`
	`{`
	`  g_Text = "x is unknown";`
	`}`

The switch-structure

- Remarks:
  - The variable has to be an *int*, or another type that can be treated as an *int* (eg. TCHAR).
  - The switch executes code starting from the first matching *case*.
  - The *break;* statements are optional. If a case block isn't terminated by a *break*, the code will keep on executing lines from the next case, until it hits a *break*, or the end of the switch.
  - The *default:* part is optional. This is where you write the code that needs to be executed if none of the given values match.

The ? : operation

- A control-expression.
- Pros: can be used in some cases, to write a selection as an inline structure.
- Example:

```
int b = ( a == 10) ? 5 : 8;
```

Boolean expression

when true, b = 5

when false, b = 8

The ? : operation

- Remarks:
  - The first operand of the ?: - operation is a *boolean expression.*
  - Het result of the ?: - operation depends on the result of the first operand:
    - If it is *true*, then the result is the part after the ?
    - If it is *false*, then the result is the part after the :

Topics

- Selections
- Enumerations

Scoped enumerated types (opsomming)

- *Enumerated types* are types that are defined with a set of custom identifiers(names), known as *enumerators*, as possible values. Objects of these *enumerated types* can take any of these enumerators as value.
- Their syntax is:

```
enum class Type_name
{
   value1, value2, value3,...
} enumeration object names;
```

Scoped enumerated types (opsomming)

- Examples: Colors, Directions,…
- Declaration example:

```
enum class DirectionState
{
   left, right, up, down
};

DirectionState g_Direction1{ DirectionState::left };
```

Scoped enumerated types (opsomming)

```
enum class Direction{left, right, up, down};
Direction myDirection { Direction::left};
switch (myDirection)
{
    case Direction::left:
       //do stuff
       break;
    case Direction::right:
       break;
    default:

}
```

Scoped enumerated types (opsomming)

```
enum class Direction
{
    left,
    right,
    up,
    down
};
```

The enumerators are actually integer numbers that are automatically assigned to each of them.

Scoped enumerated types (opsomming)

```
enum class Direction
{
    left = 0,
    right = 1,
    up = 2,
    down = 3
};
```

The enumerators are actually integer numbers that are automatically assigned to each of them.

You can manually choose the integer value.

Enumerators can be typecasted to integers if needed.

Scoped enumerated types (opsomming)

Enumerators can be typecasted to integers if needed.

Example:

```
enum class Direction
{
    left = -1,
    right = 1
};
Direction dir { Direction::left };
newPosX += int(dir) * 50;
```