Variables 2

Variables 2

- Simple functions
- modulo operator
- typecasting / integer division
- prefix vs postfix

Functions

- Functions are used to split up code in to different parts to enhance readability and for reusability.
- They have many more possibilities, but here we will use it's simplest form only to avoid huge code chunks making the code more readable.
- Three steps:
  - Write the function prototype of declaration
  - Write the function body or definition
  - Call the function
- C++ Coding standards – rule 20: Avoid long functions.
- This is just the basic usage, more on functions later!

DIGITAL ARTS & ENTERTAINMENT

Functions example usage

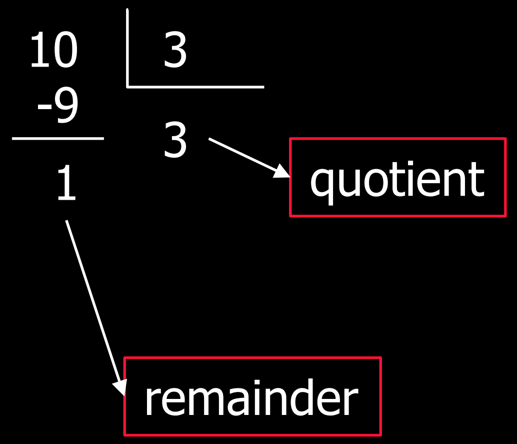
```cpp
#include <iostream>
void PrintNumbers(); // step 1: forward function declaration
int main()
{
    std::cout << "Hello World!\n";
    PrintNumbers(); // step 3: call the function
    std::cout << "Enter an angle in radians:";
}

void PrintNumbers() // step 2: function definition
{
    std::cout << "0123456789" << '\n';
}
```

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

The modulo operator %

- Finds the remainder of integer division of one number by another,
- See long division
- Works only on integer data types



The modulo operator %

- The modulo operator % finds the remainder of integer division of one number by another.
- Examples:

0%3	0
1%3	1
2%3	2
3%3	0
4%3	1
5%3	2
6%3	0

The modulo operator %

- The modulo operator % finds the remainder of integer division of one number by another.
- Examples:

0%3	0
1%3	1
2%3	2
3%3	0
4%3	1
5%3	2
6%3	0

0%2	0
1%2	1
2%2	0
3%2	1
4%2	0
5%2	1
6%2	0

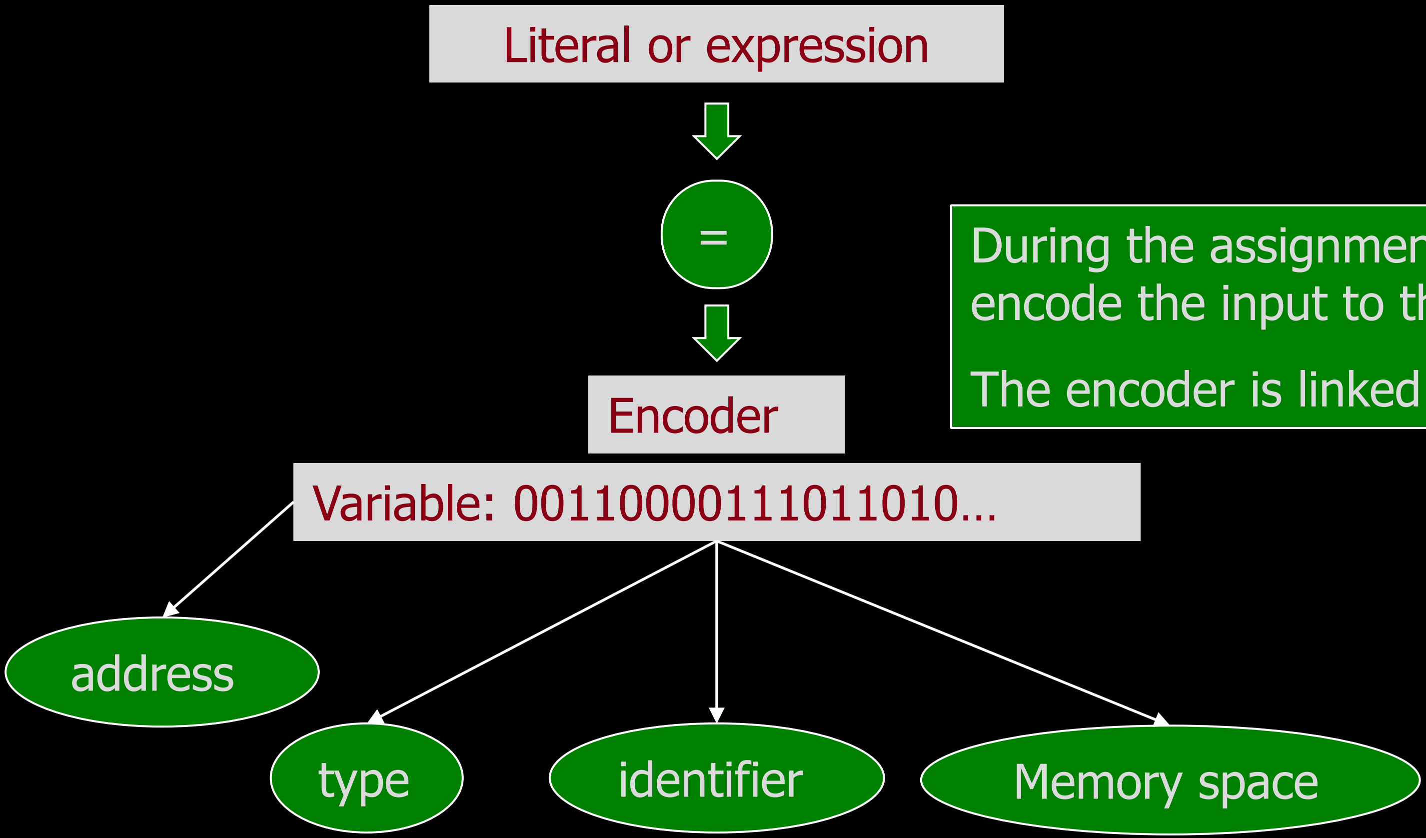
The assignment operator (=)

```
a = 10;
```

$$\underbrace{b = \overbrace{10 - 8 + 4 * 8}^{\text{expression}};}_{\text{statement}}$$

```
c = 3.1415926535;
```

The assignment operator (=)

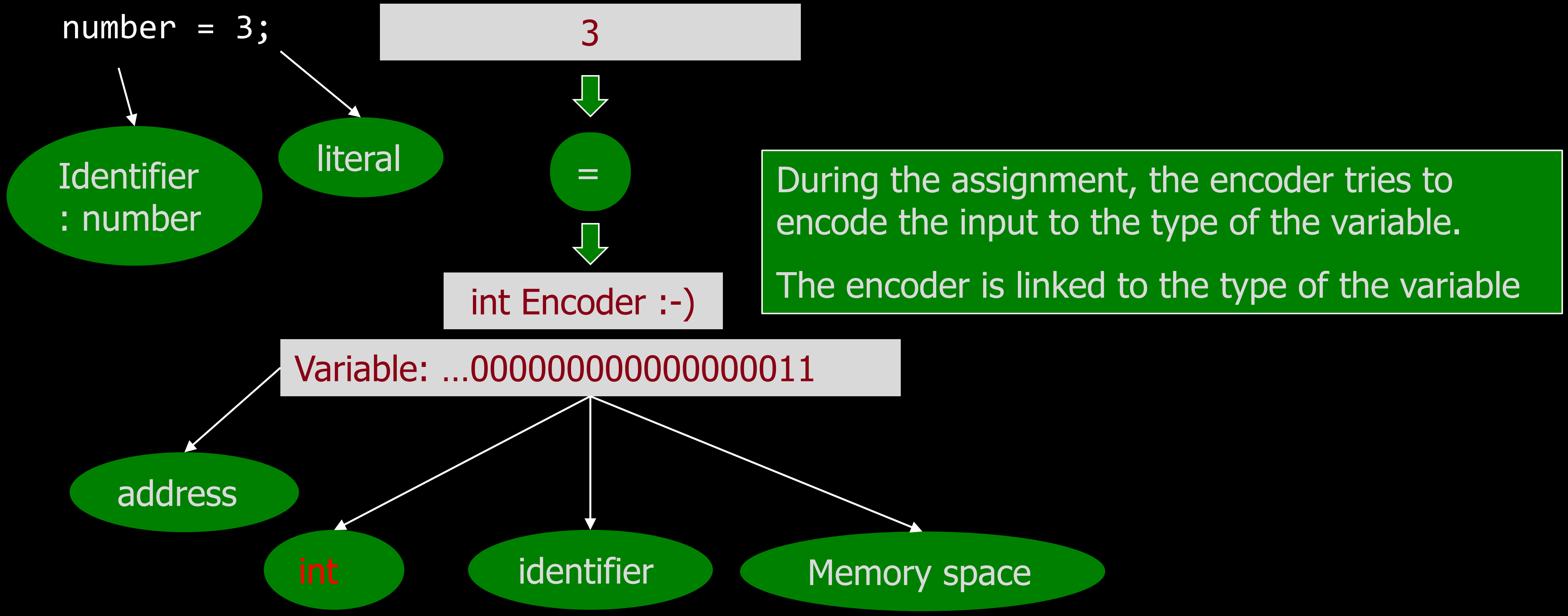


During the assignment, the encoder tries to encode the input to the type of the variable.

The encoder is linked to the type of the variable

The assignment operator (=)

```
int number{};
number = 3;
```



The assignment operator (=)

```
int number{};

number = 3.14;
```

3.14

=

int encoder :-(

Identifier
: number

literal

Variable: ??????????????????????

address

int

identifier

Memory s

During the assignment, the encoder tries to encode the input to the type of the variable.

The encoder is linked to the type of the variable

Compiler warning: the literal 3.14 can not be converted to variable of type int.

warning C4244: conversion from 'double' to 'int', possible loss of data

The assignment operator (=)

```
int number{};
number = 3.14;
```

3.14

=

int encoder :-(

Variable: ...0000000000000000011

Identifier : number

literal

address

int

identifier

Memory s

The encoder is only capable to encode the integral part of the number. This causes a warning, the fractional part is lost.

Compiler warning: the literal 3.14 can not be converted to variable of type int.

warning C4244: 'initializing' : conversion from 'double' to 'int', possible loss of data

The assignment operator (=)

```
int number{};
number = 42;
```

l-value: type is int

r-value: type is int

both must be same type

The assignment operator (=)

```
int number{};
number = 3.1415;
```

l-value: type is int

r-value: type is double

Operands are not same type > Not ok

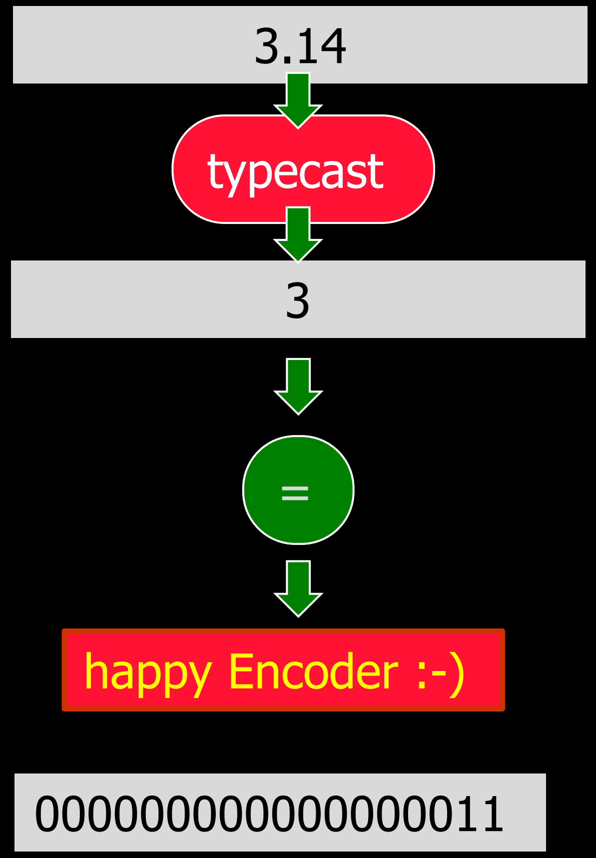
- compiler problem
- decimal part is lost
- number will contain the value 3

Explicit typecasting solves the warning

DIGITAL ARTS & ENTERTAINMENT

Typecasting: explicit type conversion

“Tell the compiler you know what you are doing”



Typecasting is explicit type conversion:

- A variable is converted to another type
    - No compiler warning
    - Decimal part is still lost
    - Number will contain the value 3

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

Typecasting: explicit type conversion

```
int i{};
float f{ 3.1415f };

//C-style typecasting
i = (int)f;

//Functional style typecasting (preferred)
i = int(f);
```

C++ is a strong-typed language:
Many conversions, specially those that imply a different interpretation of the value, require an explicit conversion, known in C++ as type-casting.

Implicit type conversions

- Are automatically performed when a value is copied to a compatible type = when no information is lost during the conversion.
- Examples:

```
float f{3.1415f};
double d{ f };
int a{'A'};
double e{ 10 };
```

Implicit type conversions

- Promotion
  - When a smaller type is converted to a larger type.
  - Guarantees the exact same value
  - No compiler messages

```
float f{3.1415f};
double d{ f }; -> promotion from float to double

int i{ 45 };
double d2{ i }; -> promotion from integer to double
```

DIGITAL ARTS & ENTERTAINMENT

Implicit type conversions

- Promotion examples

```
char c{ 'A' };
int i{ c }; -> c is promoted from char to int
```

- Math with char types promoted to int

```
char c{ 'A' };
std::cout << c << '\n'; -> prints A
std::cout << c + 0 << '\n'; -> prints 65
std::cout << char(c + 0) << '\n'; -> typecasting, prints A
```

Implicit type conversions

- No Promotion
  - When a type is converted to a not compatible type.
  - No guarantee to produce the exact same value.
  - The compile may signal the problem with a warning.

```
float f1{ 3.1415f };
int i1 { f1 }; -> warning (float to integer)
float f2{ 3.1415 }; -> warning (double to float)
```

Implicit type conversions

- Assignment examples

```
char c{ 'A' };
// type of literal -15 is integer, c is a char
c = -15; -> ok, -15 is in range of [-128, 127]
c = 150; -> NOT ok, warning: '=': truncation of constant value
```

Intel i386 cpu and i387 math co processor

DIGITAL ARTS & ENTERTAINMENT

Integer division

- whole or integer division
  - When a operator is used on integer operands, the result is an integer number.
  - Example: 5 / 4

operands

`4.0 + 8`

operator

- floating point division:
  - When at least one operand is a floating point number, the other is promoted, and the result is a floating point value.
  - Example: 5 / 4.0

Integer division

Example:

```
float aspectRatio { float(1280 / 720) };
```

- Value of aspectRatio?

Integer division

Example:

```
float aspectRatio { float(1280 / 720) };
```

- Value of aspectRatio?

1.0f

Integer division

Example:

```
float aspectRatio { float(1280 / 720) };
```

- Integer division
- The expression 1280 / 720 results in the value 1 and 1 is used to initialize the floating point variable.
- No warning (!)

Integer division

Example:

```
float aspectRatio { 1280.0 / 720 };
```

- Floating point division
- The expression 1280.0 / 720 results in the value 1.777 and is used to initialize the floating point variable.
- Warning (!) > why?

Integer division

Example:

```
float aspectRatio { 1280.0 / 720 };
```

- Floating point division
- The expression 1280.0 / 720 results in the value 1.777 and is used to initialize the floating point variable.
- Warning (!) conversion from double to float

Integer division

Example:

```
float aspectRatio { 1280.0f / 720 };
```

- Floating point division
- The expression 1280.0 / 720 results in the value 1.777 and is used to initialize the floating point variable.
- Everything ok now.

Integer division

Example:

```
float aspectRatio { float(1280) / 720 };
```

- Floating point division
- The expression 1280.0 / 720 results in the value 1.777 and is used to initialize the floating point variable.
- Everything ok now.

Typecasting: attention

- What is the value of a after this line of code is executed?

```
double a { 5 / 2 + 5.0 / 2 };
```

Typecasting: attention

- What is the value of a after this line of code is executed?

```
double a { 5 / 2 + 5.0 / 2 };

          2    + 5.0 / 2

          2    +   2.5     -> 4.5
```

Implicit type casting has the highest priority

DIGITAL ARTS & ENTERTAINMENT

Resulting type of an expression

```
5 / 4             -> int
5.0 / 4           -> double
5 / 4.0f          -> float
int(5.0) / 4      -> int
```

If at least one of the 2 operands is a floating point type, the evaluation of the expression will be a floating point type.

If the type of both of the 2 operands is integer, the evaluation of the expression will be an integer.

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

Resulting type of an expression

```
int number1{ 4 }, number2{ 5 };
double number3{ 3.1415 };
int result { number1 + number2 + number3 };
```

Compiler error

Resulting type of an expression

```
int number1{ 4 }, number2{ 5 };
double number3{ 3.1415 };
int result { int( number1 + number2 + number3 ) };
```

type is int

typecasted expression: type is int

Typecasting removes the decimal part

Resulting type of an expression

```
int number1{ 4 }, number2{ 5 };
double number3{ 3.1415 };
int result { int( 4 + 5 + 3.1415 ) };
```

type is int

typecasted expression: type is int

Typecasting removes the decimal part

Resulting type of an expression

```
int number1{ 4 }, number2{ 5 };
double number3{ 3.1415 };
int result { int( 12.1415 ) };
```

type is int

typecasted expression: type is int

Typecasting removes the decimal part

Resulting type of an expression

```
int number1{ 4 }, number2{ 5 };
double number3{ 3.1415 };
int result { 12 };
```

type is int

typecasted expression: type is int

Typecasting removes the decimal part

Unary operators

- Prefix vs postfix

The increment (++) and decrement (--) operator

```
number = number + 1;
number = number - 1;
```

```
number++; or ++number;
number--; or --number;
```

The increment (++) and decrement (--) operator

```
number = number + 1;        number++; or ++number;
number = number - 1;        number--; or --number;
```

Prefix: ++number; or Postfix: number++;

```
double numberOfPieces{ 10 };
double pricePerPiece{ 5.00 };
```

What is the difference?

```
total = ++numberOfPieces * pricePerPiece;
total = numberOfPieces++ * pricePerPiece;
```

The increment (++) and decrement (--) operator

```
number = number + 1;
number = number - 1;
```

```
number++; or ++number;
number--; or --number;
```

Prefix: ++number; or Postfix: number++;

```
double numberOfPieces{ 10 };
double pricePerPiece{ 5.00 };
```

What is the difference?

```
total = ++numberOfPieces * pricePerPiece; // total: 55
total = numberOfPieces++ * pricePerPiece; // total: 50
```

References

- http://www.cplusplus.com/doc/tutorial/variables/
- http://www.learncpp.com/cpp-tutorial/11-structure-of-a-program/