

Lecture 2: Value Added Syntax

Bart Iver van Blokland

PSA

- We want to see whether we should adjust how many student assistants are available throughout the week
- Please let us know which times you would like to drop by to get help from our student assistants
 - Note: this is not a binding registration!



<https://nettskjema.no/a/585338>

Last time..

- Project configuration
- Program compilation
- The main function
- Writing to the terminal using cout

Last time..

- Project configuration
 - When you use the “create project” command in VS Code, the Meson tool will run and configure your project
- Program compilation
- The main function
- Writing to the terminal using cout

Last time..

- Project configuration
 - When you use the “create project” command in VS Code, the Meson tool will run and configure your project
- Program compilation
 - The process of converting code to binary commands that your computer’s processor can understand and run
- The main function
- Writing to the terminal using cout

Last time..

- Project configuration
 - When you use the “create project” command in VS Code, the Meson tool will run and configure your project
- Program compilation
 - The process of converting code to binary commands that your computer’s processor can understand and run
- The main function
 - The starting point of your program
- Writing to the terminal using cout

Last time..

- Project configuration
 - When you use the “create project” command in VS Code, the Meson tool will run and configure your project
- Program compilation
 - The process of converting code to binary commands that your computer’s processor can understand and run
- The main function
 - The starting point of your program
- Writing to the terminal using cout
 - Useful for annoying your future self =)

Today

Basics of the C++ Language

- Variables and data types
- Language elements



You are challenged by Ekans!



Ekans sent out Python!



Go CPLUSPLUS!

PYTHON

Lv31



CPLUSPLUS

Lv79



What will CPLUSPLUS do?

> FIGHT LANGS

ITEM RUN

PYTHON

Lv31



CPLUSPLUS

Lv79



What will CPLUSPLUS do?

> AVAILABLE LIBRARIES
TIGHT CONTROL
SPEED

PYTHON

Lv31



CPLUSPLUS

Lv79



CPLUSPLUS used AVAILABLE LIBRARIES!

PYTHON

Lv31



CPLUSPLUS

Lv79



CPLUSPLUS used AVAILABLE LIBRARIES!

It's not very effective..

PYTHON

Lv31



CPLUSPLUS

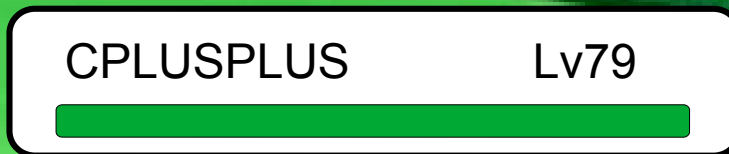
Lv79



What will PYTHON do?

> FIGHT LANGS

ITEM RUN



What will PYTHON do?

> NO NEED TO COMPILE
AVAILABLE LIBRARIES
PIP INSTALL

PYTHON

Lv31



CPLUSPLUS

Lv79



PYTHON used NO NEED TO COMPILE!

PYTHON

Lv31



CPLUSPLUS

Lv79



PYTHON used NO NEED TO COMPILE!

It's not very effective..

PYTHON

Lv31



CPLUSPLUS

Lv79



What will CPLUSPLUS do?

> FIGHT LANGS

ITEM RUN

PYTHON

Lv31



CPLUSPLUS

Lv79



What will CPLUSPLUS do?

AVAILABLE LIBRARIES

TIGHT CONTROL

> SPEEEEEED

PYTHON

Lv31



CPLUSPLUS

Lv79



CPLUSPLUS used SPEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEED!

PYTHON

Lv31



CPLUSPLUS

Lv79



It's SUPER effective!!



PYTHON has fainted!

Last time..

- Project configuration
- Program compilation
- The main function
- Writing to the terminal using cout

Addendum: print() / println()

- Alternative to cout for writing to the terminal
 - New in C++23
 - println() is the same as print() but also creates a new line

```
// Practically all C++ code written to date  
cout << "Hello there!" << endl;
```

```
// New way  
println("Hello there!");  
println("You have {} unread messages", 10);  
println("The number {1} is less than {0}", 5, 4);  
print("You can ");  
print("also write using ");  
print(" statements.");  
// Not allowed: print(42);  
// Do instead: print("{} ", 42);
```

Today

Basics of the C++ Language

- Variables and data types
- Language elements

Today

Basics of the C++ Language

- **Variables and data types**
- Language elements

Variables

```
int variableName;
```



Type (this one is an integer)



Name

```
int variableName = 5;
```



Can optionally be initialised to a value
(and you really should)

- Variables allow you to store values
 - A variable is a “space” that stores a single value
 - Stored values remain the same until updated
- Used all the time!
- Work the same as in Python

Variables

```
int variableName;  
cout << "The value of variableName is: " << variableName << endl;
```

- Unlike Python, it is possible to create a variable without assigning a value to it
- The contents will be undefined!
 - Best practice: always initialise your variables!
 - The compiler will warn you about this:

```
[6/13] Compiling C++ object program.p/main.cpp.o  
../main.cpp: In function 'int main()':  
../main.cpp:9:17: warning: 'variableName' is used uninitialized [-Wuninitialized]  
    9 |         cout << variableName << endl;  
      |         ^~~~~~
```

Variables

```
int variableName = 10;  
int anotherVariable{6};
```

```
variableName = 5;
```

```
variableName = anotherVariable + 3;
```

```
std::cout << variableName << endl;  
std::cout << anotherVariable << endl;
```

- Variables should be initialised with a value to avoid weird errors
- The = operator overwrites the value of the variable
- The values of variables can be used in calculations
- Prints:
9
6
The contents of variables are thus independent!

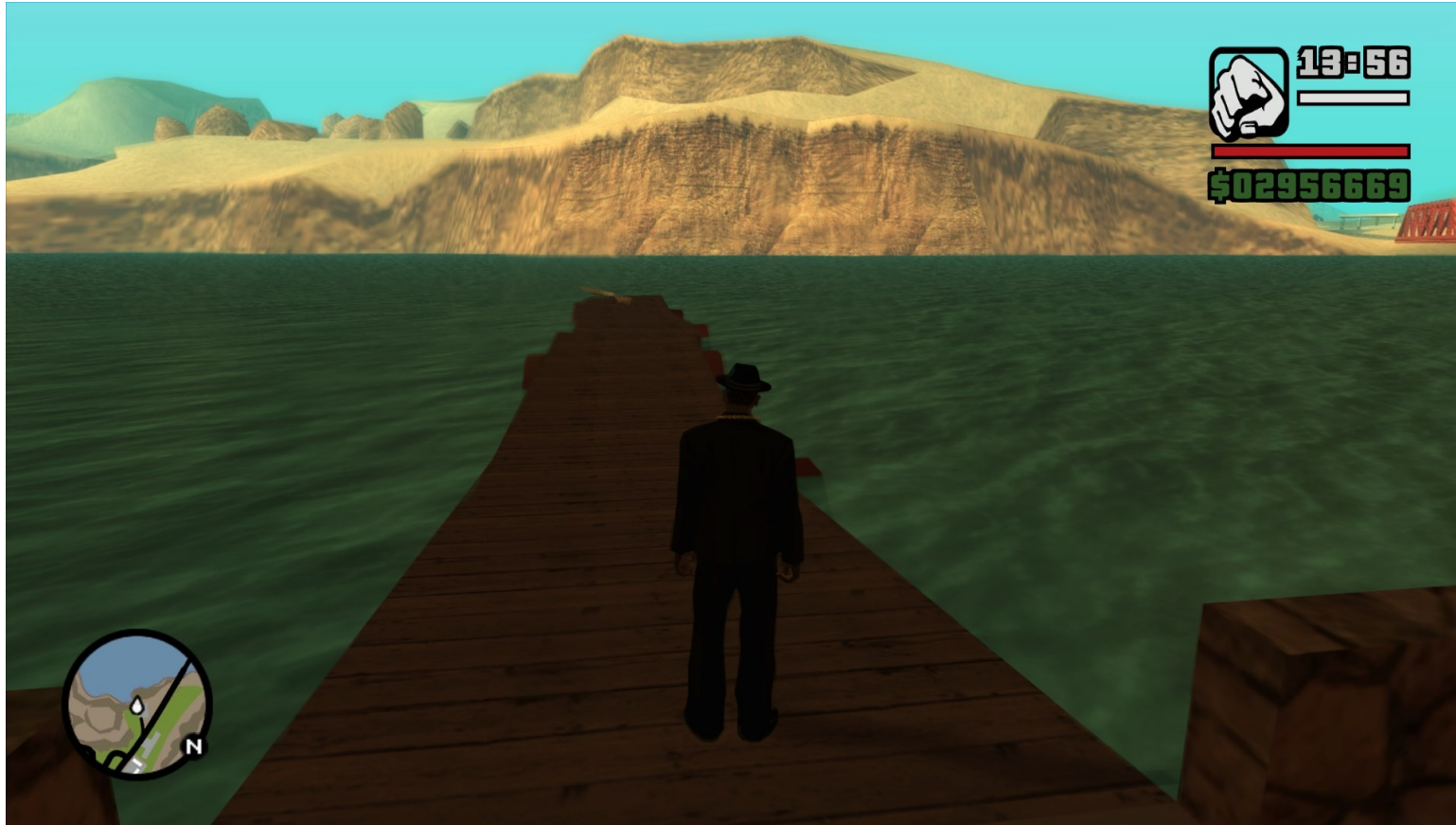
Uninitialised memory: what can happen?



Source:



Uninitialised memory: what can happen?



Source:



Uninitialised memory: what can happen?

- The storage space for function variables tends to get reused.
- The developers forgot to specify the size of the plane's wheels in the configuration file
- Because the variables for these were not initialised, they were set to whatever last touched them
 - Used to be a variable storing a similar value
 - Windows update changed a function used by the game, which modified that memory location to a massive value instead.

Source:



Data types

```
int variableName;
```



Name

Data type (this one is an integer)

- Why do we care about data types?
- Which data types exist?
- How do we choose which one to use?

Data Types

Category	Data Type
Integers	
Real numbers	
Miscellaneous	

Data types: integers

`int` variableName;
↑ ↑
Integer data type Name

- Stores whole numbers (0, 42, -273, 1337, ..)
- Risk: only a certain number of binary digits (bits) can be stored.
 - When the highest possible value is exceeded, it loops around to the lowest possible value, and vice versa

The effects of data types: Demonstration

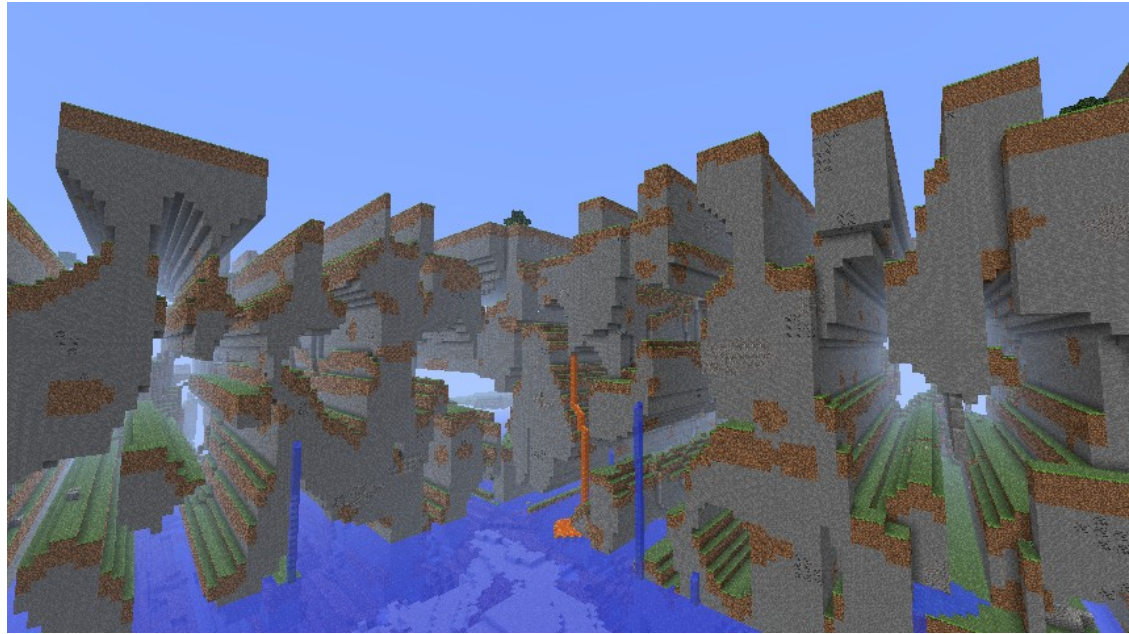
The effects of data types: Minecraft's far lands

The far lands in beta 1.7.3 were caused by an integer overflow.

The terrain generator multiplies all block coordinates by approximately 171.

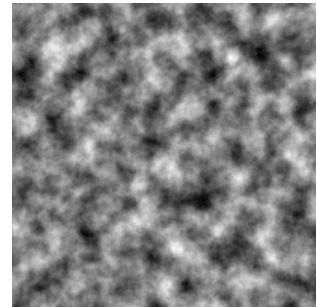
A part of this algorithm converts that number into an integer.

After 12,550,224 blocks this causes an integer overflow!



The far lands

Perlin noise, used
in early versions of
minecraft



Data types: real numbers

```
double variableName;
```

Format of a 32-bit float:

sign (1 bit)	exponent (8 bits)	mantissa (23 bits)
0	10000101	000101000000000000000000

- Sign: 1 if number is negative, 0 if positive
- Exponent: which power of 2 the number lies between (e.g. 2, 4, 8, 16, 32)
 - Negative for numbers between -1 and 1
- Mantissa: represents how far the number lies between the selected powers of 2
 - Linearly subdivided
 - 32-bit float: subdivided into 2^{23} numbers

Data types: real numbers

```
double variableName;
```

- Stores real numbers (3.14, -273.15)
- Risk: after each calculation, the result is *rounded* to the nearest representable number
 - Causes noise
 - Try to keep numbers close to 0 if possible
 - Use **double** if precision matters
 - Avoid computations combining large and small numbers
- Example: 32-bit **float** can only represent increments of 0.25 for numbers between ~2 100 000 and ~4 200 000

Data types: real numbers

```
double variableName;
```

- Note: adding two floating points together also adds their errors
 - Usually not a problem unless you do this many times
- Note: dividing a real number by 0 creates a special value that can show up as Not A Number (“nan”) or Infinity (“inf”).

The effects of data types: Minecraft's far lands

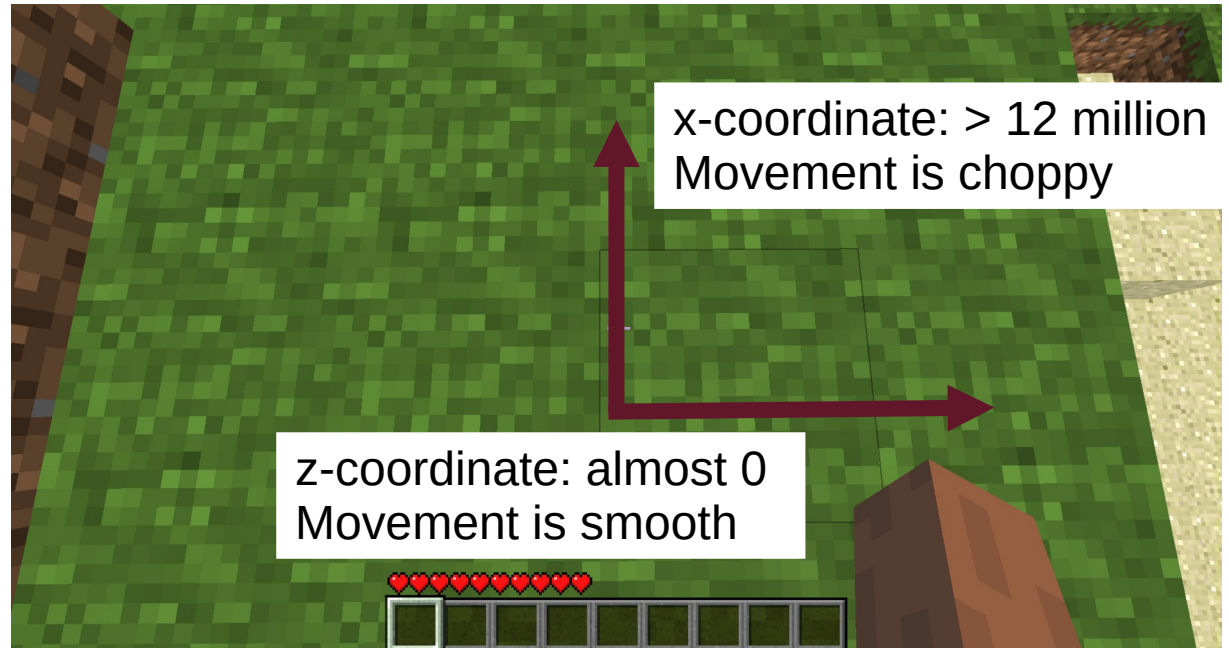
Movement was choppy in one direction but not in the other.

Why?

The higher a floating point number gets, the larger the distance between numbers they can represent.

The game stores coordinates as a 64-bit double, but at some point converted them to a 32-bit float

Around 12 million, 32-bit floats can only represent integers



Data Types

Category	Data Type
Integers	char
	int
Real numbers	
	double
Miscellaneous	bool
	string
	enum class
	struct and class
	void (no data type)

These are the data types that are usually enough for most programs.

Data Types

Category	Data Type
Integers	char
	unsigned char
	short
	unsigned short
	int
	unsigned int
	long long
	unsigned long long
Real numbers	float
	double
Miscellaneous	bool
	string
	enum class
	struct and class
	void (no data type)

Data Types

Category	Data Type	Number of bytes
Integers	char	1
	unsigned char	1
	short	2
	unsigned short	2
	int	4
	unsigned int	4
	long long	8
	unsigned long long	8
Real numbers	float	4
	double	8
Miscellaneous	bool	usually 1
	string	variable
	enum class	usually 4
	struct and class	variable
	void (no data type)	not applicable

One byte is 8 bits

One bit is a zero or one

Example: 01001011

Number of bytes influences the number of values a type can store, and how much space it occupies

Data Types

Category	Data Type	Lowest value	Highest value
Integers	char	-128	127
	unsigned char	0	255
	short	-32,768	32,767
	unsigned short	0	65,535
	int	-2,147,483,648	2,147,483,647
	unsigned int	0	4,294,967,295
	long long	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
	unsigned long long	0	18,446,744,073,709,551,615
Real numbers	float	$-\infty$	∞
	double	$-\infty$	∞
Miscellaneous	bool	not applicable	not applicable
	string	not applicable	not applicable
	enum class	not applicable	not applicable
	struct and class	not applicable	not applicable
	void (no data type)	not applicable	not applicable

Data Types

4 questions:

- Why are there different integer and real types?
- Why are there signed and unsigned integers?
- What happens if the integer values go out of range?
- How do I decide what data type to use?

Data Types

4 questions:

- **Why are there different integer and real types?**
 - We sometimes work with data which uses a specific number of bytes per value.
 - Smaller data types may result in better performance
- Why are there signed and unsigned integers?
- What happens if the integer values go out of range?
- How do I decide what data type to use?

Data Types

4 questions:

- Why are there different integer and real types?
- **Why are there signed and unsigned integers?**
 - Modern processors support both
 - Use signed for day-to-day computations
 - Use unsigned for manipulating bits, or to show a value is always positive
- What happens if the integer values go out of range?
- How do I decide what data type to use?

Data Types

4 questions:

- Why are there different integer and real types?
- Why are there signed and unsigned integers?
- **What happens if the integer values go out of range?**
 - The value “loops around”, so probably bad things!
- How do I decide what data type to use?

Data Types

4 questions:

- Why are there different integer and real types?
- Why are there signed and unsigned integers?
- What happens if the integer values go out of range?
- **How do I decide what data type to use?**
 - Need an integer? You usually want to use `int`
 - Need a real number? You usually want to use `double`
 - `bool` and `string` should be obvious

Data Types

One note regarding overflow:

- The C++ standard considers overflow behaviour of signed integers to be undefined behaviour
- The last machine that did not use signed integers in their hardware was made in the 1980's
- We therefore ignore the opinion of the C++ standard in this course

Quiz: which data type?

char

int

long long

double

bool

string

- Last name
- Number of seconds a program has run
- A temperature
- Fødselsnummer
- Whether an assignment has been approved
- The average of two numbers between 0 and 1 000 000
- Balance on a bank account

Today

Basics of the C++ Language

- Variables
- Data types
- **Type casting**
- Operators
- Text input
- If statements
- Loops
- Functions

(this will be all you need to complete assignment 1)

Type casting

- Casting is conversion of data types
 - This can lead to precision or data loss, but the compiler can warn you
- Types are converted implicitly if no accuracy is lost
 - Rule of thumb: if a type uses more bytes, you can safely convert to it
 - For example, `char` to `int` or `float` to `double`

- Types can be converted explicitly using `static_cast<>()`:

```
int anInteger = 5;  
float floatValue = static_cast<float>(anInteger);
```

- For converting between numeric types, you can also use the type as a function:

```
int anInteger = 5;  
float floatValue = float(anInteger);
```


The name «casting» comes from metallurgy!



Type casting

- Why explicit type conversion when much of it happens implicitly?
 - Types can not always be converted implicitly
 - Explicitly converting a type shows the conversion is intentional
 - And a loss of accuracy is acceptable
 - For numeric types, some calculations require type conversions to produce correct results
- Converting numbers to and from strings is a bit different.
We'll look at those in the next lecture.

Today

Basics of the C++ Language

- Variables
- Data types
- Type casting
- **Operators**
- Text input
- If statements
- Loops
- Functions

(this will be all you need to complete assignment 1)

Operators

- Roughly two main categories (that we use in this course):
 - Arithmetic operators: compute stuff
 - Logic operators: compare stuff

Arithmetic operators

Calculation

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
^	XOR

Assignment

Operator	Description
+=	Increment by
-=	Decrement by
*=	Multiply by
/=	Divide by
%=	Remainder of
=	Assignment
++	Increment by 1
--	Decrement by 1

Examples

```
int sum = 5 + 3;  
cout << sum << endl; // 8  
  
sum++;  
cout << sum << endl; // 9  
  
sum *= 3;  
cout << sum << endl; // 27
```

Operators

- The resulting data type of an operation depends on its operands

// If one of both operands is a float/double, the result is too

`double + double = double`

`double + int = double`

`int + double = double`

`int + int = int`

// For similar integer types, signed and unsigned becomes unsigned

`int + unsigned int = unsigned int`

// If one of the two operands uses more space, the result does too

`int + char = int`

`float + double = double`

// Comparison operators always result in a bool

`int < double = bool`

Operators

- Let's apply these!

`double + float = ?`

`char + short = ?`

`unsigned long long + int = ?`

`int / int = ?`

`double / int = ?`

Logic operators

Value comparison

Operator	Description
<	Is less than
<=	Is less than or equal to
==	Is equal to
!=	Is not equal to
>=	Is greater than or equal to
>	Is greater than

Examples

```
bool lessThan = 5 < 4;  
cout << lessThan << endl; // 0
```

```
bool notEqual = 3 != 9;  
cout << notEqual << endl; // 1
```

Boolean

Operator	Description
!	Boolean NOT
&&	Boolean AND
	Boolean OR

```
bool inverse = !(3 > 2);  
cout << inverse << endl; // 0
```


Operators

- C++ does not have Python's `//` operator
 - When both operands are integers, integer division is used. Otherwise, always float division.
- A float in Python is always 64-bit. In C++ that would be called a double.
- The order in which operators are evaluated follows algebraic rules.
- Good practice: do not use the `==` and `!=` operators with floating point numbers.

Operators

- Common mistakes:
 - Using `=` instead of `==`
 - `0 <= a <= 10`
 - Use `0 <= a && a <= 10` instead