UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 3: Variables and Data types

mihasan@liverpool.ac.uk

Previous lecture

- C++ components were discussed, which consist of keywords, syntax rules, and developers statements.
- The role of pre-processor directives and header files in a programme was discussed.
- The libraries `<iostream>, <string>, <fstream>, and <chrono>` were discussed.
- The types of errors where discussed and examples of such errors were given.

This lecture

- What is covered in this lecture?

  1. Data types of simple variables 2. Arrays 3. Operators and precedence

- Why it is covered?

  1. Variables are the basic building block of any code/programme.

  2. Operators are the foundation of any programming language.

- How are topics covered in this lecture:

  2 source codes

Variables and Data types

- Variables are *volatile (or temporary)* storage for keeping data that a program uses, where the storage has a certain size and location in the memory.
- The syntax of defining a variable is as follows:

```
DataType VariableName = VariableValue;
```

Variable declaration: tells the complier the variable exists

Variable Initialisation: sets the variable’s value

- Variable's declaration and initialisation can be independent steps.

```
DataType VariableName;

VariableName = VariableValue;
```

Variables and Data types

- There are multiple ways to declare and initialise variables:

```
#include <iostream>


int main () {

   int a = 2;  // initialization using “=“ operator

   int b (3); // initialization using parentheses (not recommended)

   int c {2}; // initialization using braces

   int d , e = 5, f = 4; //multiple declaration and initialization

}
```

Variables and Data types

- In C++ the particular type of each data must be declared before it is used.
- The most used types in C++ are:

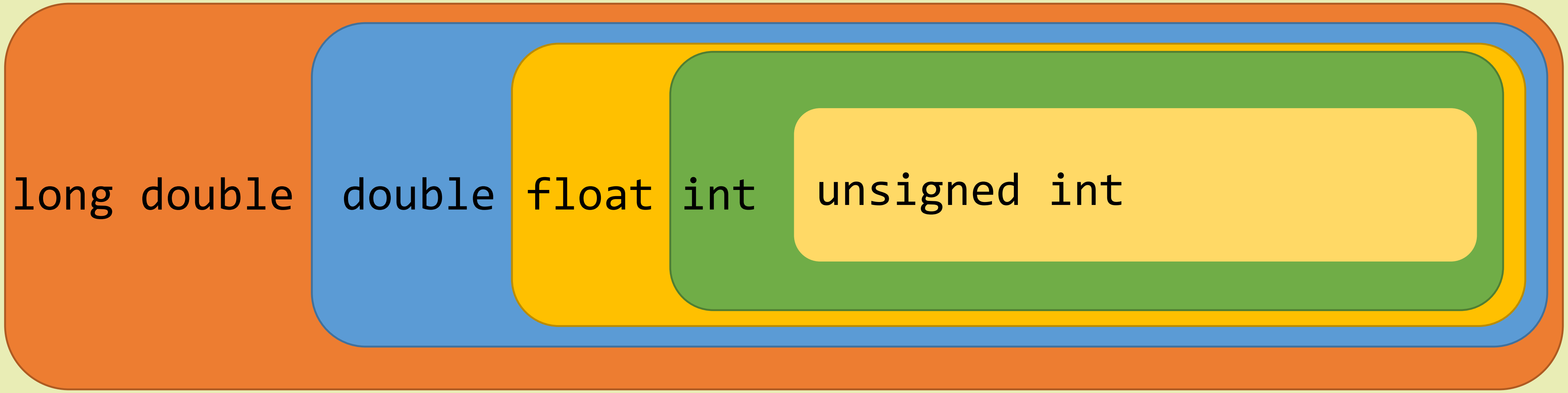
Data type	Use	Example
`int`	Numeric variables without decimal	`int a = 5; int b = -1;`
`bool`	Boolean variables (True or False)	`bool a = true; bool b = 0;`
`char`	Characters and symbols variables	`char a = ‘s’; char b = ‘$’;`
`double`	Most general numerical variables	`double a = 2.3; double b =-2.6;`
`auto`	Automatically determines the type	`auto a = ‘s’; auto b = 3;`

- Full list of variables in C++ is here: https://en.cppreference.com/w/cpp/language/types
- If the value of the variable is constant, it can be made a constant variable:

```
const int PI = 3.141; // Must be initialised when declared
```

Variables and Data types

- Many of datatypes in C++ are special cases of others:



- QUESTION: why are very specific datatypes (such as `unsigned int`) still in use?
- Two viewpoints: https://www.youtube.com/watch?v=wvtFGa6XJDU

Variables and Data types

- Two general reasons for using very specific datatypes:

1. Bitwise operations.

2. Improve the code's readability.

- Go to L3D1.cpp

Practical note:

- Don't use unsigned types for arithmetic operations.
- For scientific calculations use "`double`" instead of "`float`".
- Minimise the use of "`auto`" to make it easier to troubleshoot your code.

Variables Naming

- C++ is case sensitive!
- Large software projects have conventions for naming variables, to make the code easier to read and debug.
- In this module, everyone should follow the “Google C++ Style guide” https://google.github.io/styleguide/cppguide.html (mention that in your cover letter!).
- Some examples:

Variable	Chosen name
Number of students	number_of_students ; numberofstudents
Switch status	is_switch_on; isswitchon
Number of days in a week	kDaysInAWeek

Enumeration

- Enumeration is a user defined data type where a set of values is specified for a variable and the variable can only take one out of a small set of possible values.
- The Enumeration has to be declared before being used.

```
enum UserDefinedDatatype {kvalue_1,kvalue_2,..,kvalue_last};
```

- Example:

```
enum Season {autumn, winter, spring, summer};

Season currnet_season; // declaration

current_season = autumn ; // It has to be one of the 4 options
```

Scope of variables

- In C++ variables can be declared anywhere within the code.
- The scope of a variable is defined as the extent of the code within which the variable can be accessed or worked with, which is typically the code block where the variable is declared (between {}).
- When a variable goes out of scope it is automatically deleted.
- Example:

```
int main () {

 int a = 2;

 { int a = 5 ; // Declaration in the new scope
 }
}
```

Namespaces

- A namespace is a declarative region that provides a scope to the identifiers inside it.
- They allow the developer to access a scope from outside it (a name of the scope).
- An entire code can access a namespace by including "**using namespace ..**".
- Example:

```
namespace name_space_1
{       int x = 1; }

namespace name_space_2
{       int x = 5; }

cout << name_space_1::x; // What is the output?
```

Practical note: Do not use "using namespace" when working on large codes

Global variables

- Global variables are variables that can be accessed from any part of the code.
- They are declared before `main` () and they have to be declared outside any scope.
- Global variables are only deleted when the programme terminates.
- They can be accessed directly by their name or using the scope resolution operator (::) if another local variable exists with the same name.
- Go to L3D2.cpp

Practical note:

- Do not use global variables unless there is a very good reason for that.

Operators and precedence

- Operators are symbols used to to perform specific mathematical and logical computations on operands.
- A brief list of operators: (full list http://www.cplusplus.com/doc/tutorial/operators/)

Type	Operator	Type	Operator
Arithmetic	`+, -, *, /, %`	Unary	`++, --`
Relational	`<, <=, >, =>, ==, !=`	Assignment	`=, +=, -=, *=, /=, %=`
Logical	`&&, ||, !`	Bitwise	`&, |, >>, <<, ~`

- Logical operators operate on the variable (or expression as a whole) while bitwise operators operate on a bit-by-bit basis.

Operators and precedence

- Example on Logical vs bitwise operators:

```
int a = 2, b = 1; // Remember that 2 = 10 and 1 = 01 in binary

bool c = a && b; // c = true (because neither a or b is 0)

bool c = a & b; // c = false (because 1&0 = 0&1 = 0)
```

- Assignment operators are used to assign a value to a variable.
- The operation “`a += 2`” is equivalent to “`a = a + 2`”. The same is true for other versions of this operator.

Operators and precedence

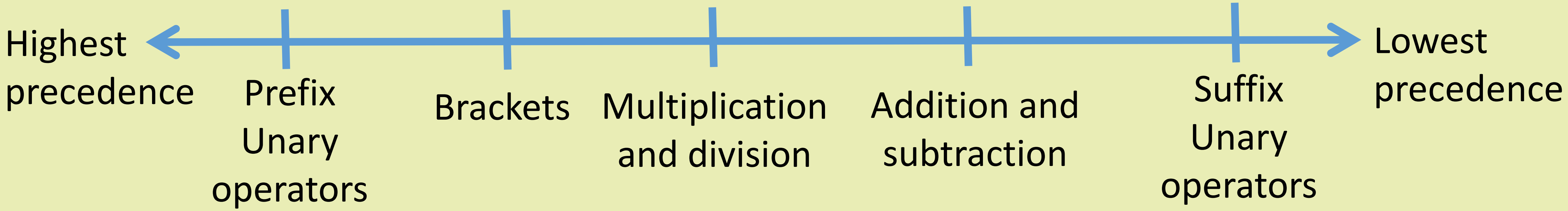
- Unary operators are operators that operate on one operand, therefore the name.
- The operation “a++” is equivalent to “`a = a + 1`”. The same is true for “`a--`”.
- If the operator is used as a prefix “++a”, “a” will be incremented <u>before</u> the operation.

  If it is used as a suffix “a++”, “a” will be incremented <u>after</u> the operation.
- Example:

```
int w{ 10 }, x{ 5 }, y{ 5 }, z{ 5 }, total{ 0 };

    x++; // increment x by one: x = x+1 = 6

   total = ++y + w; // (y+1) + w = 16

   total = z++ + w; // total = (z+w) = 15; z = (z+1) = 6
```

Operators and precedence

- Precedence is the order by which operators are performed in an expression containing multiple operators. Higher precedence operators are performed before lower precedence operators.



- Example:

```
int x = 5 + 3 * 6 - 8 / 2 ; // x = 5 + 18 - 4 = 19

int x = ((5 + 3)*6 - 8) / 2 ; // x = ( 90 - 8) / 2 = 82 /2 = 41
```

Variable casting

- Variable casting is an operator that changes the data type of the variable it operates on.
- Casting can be implicit, which is done automatically by the complier when an expression containing mixed data types is encountered. Or explicit, where the developer specifies the new data type of the variable. Implicit casting is also known as "promotion"
- Example:

```
int x = 4; double y =2.5;

int z =x*y ; // the complier converts x to double

int z = x*int(y) ; // What is the answer?
```

Practical note: Avoid implicit casting whenever possible.

Arrays

- An Array is a collection of elements of the same type stored continuous memory locations that can be individually accessed using an index.
- Syntax:

```
DataType array_name[kSize]; //  declaration of array
```

- The array's size must be a constant and must be specified at the time of declaration.

```
const int kSize=5; //  notice "const", without it is a syntax error

double Array[kSize];
```

- The first element of the array has an index 0.
- The *i*th element of an array "A" can be accessed using "`A[i-1]`".

Arrays initialisation

- An Array can be initialised in many ways:

```
int salaries[5]{32,45,28,55,65};

double box_dimensions[]={3.2,4,5.6}; // The size in this case is 3

int student_id[20]{20011011} ; // The rest 19 elements are 0

int carplates[100]={} ; //All elements are initialised to 0
```

- The naming of arrays follows the variable naming convention by Google coding style.
- The size of an array can be determined using the macro “`_countof()`”, which returns the number of elements in the array in “`size_t`” format (can be casted to integer).

Multi-dimensional arrays

- All the arrays we looked at up to this point are 1D arrays.

- Multi-dimensional arrays (matrices) can be defined similar to 1D-arrays:

```
DataType array_name[kSize1][kSize2]…[kSizeN];
```

- They can be initialised as follows:

```
double points[3][3]={{32,45,28}, // first row

                     {66,55,65}, // second row

                     {12,45,66}}; // third row
```

- Elements which were not initialised are set to zero.

Summary

- The data types of variables, their declaration and initialisation methods were discussed.
- Operators in C++ and their precedence were discussed.
- Variable casting was discussed.
- Arrays declarations and initialisations were discussed.
- Further reading: http://www.cplusplus.com/doc/tutorial/