

C++ lab 1

Recap of what you should have learned during the first lab:

1. Makefile

- **all** is the default Makefile target.
- How to define Makefile rules.
- Default unix environment variables:
 - **CXX**: C++ compiler
 - **CXXFLAGS**: C++ compiler flags
 - **LDFLAGS**: Linker flags (was not required for this lab)
- How to affect a value to a Makefile variable:
 - **:=** means affect to
 - **?=** means affect if not already defined
 - **+=** means concatenate (and define if not already defined)
- Default rules variables:
 - **\$<**: Contains first dependency name
 - **\$^**: Contains all dependency names
 - **\$@**: Contains target name
- You can define generic rules that match patterns by using **%**.
- Makefile functions **wildcard** and **pathsubst**.
- You can change file extensions with **\${varname:.ext1,.ext2}**.
- **.PHONY** rules are always executed.

2. C++ compiler

- There exist many different C++ compilers like **g++** and **clang++**.
- And supported C++ standards (C++98, C++03, C++11, C++14, C++17, ...).
- Compile a C++ source file to an object:
`${CXX} ${CXXFLAGS} -c [source.cpp] -o [target.o]`
- Link objects to create a binary:
`${CXX} ${LDFLAGS} [file1.o] [file2.o] ... -o [binary_name]`
- Some compiler options:
 - **-std=c++14**: Enables the C++14 standard.
 - **-pedantic**: Disable all non standard compiler features.
 - **-Wall**: Enables some warnings.
 - **-Wextra**: Enables even more warnings.
 - **-Werror**: Converts all warnings to errors.
 - **-W[name]**: Enables a specific warning.
 - **-Wno-[name]**: Disables a specific warning.
 - **-I[folder]**: Add this folder as an include directory to look for header files.
- You can apply the **C preprocessor** to your code to see what happens before actual compilation by using the **-E** option of **gcc**: `g++ ${CXXFLAGS} -E [filename]`

3. C++ headers

- Often have extension **.h**, **.H** or **.hpp**.
- Contains **function prototypes** and **class definitions**.
- Should be guarded by an **include guard**.
- Should not contain **using namespace [name]** in the global namespace (namespace pollution).
- System libraries are included with the **#include <file>** syntax (no **.h** extension).
- Our headers should be included with the **#include "file.h"** syntax.
- **#include** is just basic **C preprocessor directive** that will resursively copy and paste headers into your code.
- You can specify **default arguments** to function and class methods.
- Including a C library like **"time.h"** and **"stdio.h"** is deprecated in C++.
- C standard libraries are hidden in specific C++ headers like **<ctime>** and **<cstdio>**.

4. C++ sources
 - Often have extension `.cpp` or `.C`.
 - Contains function and class **implementation**.
 - In a class method implementation, **this** represents a **pointer to self**.
 - Here you can use things like `using namespace std` without any side effects.
5. Basic class understanding
 - Class members and methods.
 - Public and private visibility.
 - In C++ **struct** and **class** are basically the same, the only difference is:
 - All members contained in a **class** are by default private.
 - All members contained in a **struct** are by default public.
 - Constructors:
 - Constructors are named as the typename and have no return type in their declaration.
 - A **default constructor** is a constructor which can be called without arguments:
 - `Date();`
 - `Date(day=1, month=1, year=1900);`
 - **Initializer list**: Unless all class members have a default constructor, you have to initialize them through the initializer list. It is always more efficient to initialize members directly with this method.
 - `Date(int day, int month, int year): m_day(day), m_month(month), m_year(year) {`
 `/* insert code to check if date is valid here */ }`
 - `Trip(Date start, Date end): m_start(start), m_end(end) {}`
 - A **copy constructor** is a constructor that take a reference to the same type as a parameter.
 - `Date(Date& d);`
 - `Trip(const Trip& t);`
 - A copy constructor is required by `m_start(start)` and `m_end(end)` in the second initializer list example.
 - Do not worry about type cv-qualifier `const` and reference `&` yet, we will see what it means later.
 - **Implicit constructors**:
 - Unless any constructor is defined, an object defines an implicit default constructor.
 - Unless explicitly asked, an object also defines an implicit copy constructor.
 - We will see later that even more implicit methods are automatically generated if not removed.
 - To create an object of a given type without arguments the type has to be **default-constructible**. Default constructible means it should have a public default-constructor.
 - `Date d0, d1; /* requires that Date is default-constructible */`
 - `Date d0(1,1,1970), d1(1,12,1970); /* requires public Date(int, int, int) */`
 - This is also true for static arrays:
 - `Date dates[2]; /* requires that Date is default-constructible */`
 - `Date dates[2] = {Date(1,1,1970), Date(1,12,1970)}; /* public Date(int,int,int) */`
6. Miscellaneous
 - **You should be up to date with basic C pointers syntax** (`&`, `*`, `->`).
 - `int main(int argc, const char *argv[])` is the entry point of your program:
 - It returns an error code as an integer.
 - `argc` contains the number of arguments passed to your program.
 - The first argument is the name of the program itself.
 - `argv` contains all the passed arguments as an array of arrays of characters (`argv` is a `char**`).
 - Use `std::cout`, `std::cerr` and `std::endl` defined in header `<iostream>` to print to the screen.
 - Use `std::setprecision`, `std::setfill` and `std::setw` defined in `<iomanip>` to change how things are printed.
 - The C header `<cstdlib>` defines:
 - `EXIT_SUCCESS` and `EXIT_FAILURE` macros.
 - The `atoi` function that converts an array of characters to an integer.
 - The C header `<cstring>` defines `strcpy`.
 - The C header `<cassert>` defines `assert`.