**C++ for scientific computing by Geert Jan Bex**

<https://github.com/gjbex/training-material/tree/master/CPlusPlus>

**History of C++**

Origin of language in 1983 by Bjarne Stroustrup

C++98, C++11, C++14, C++17 (11, 14 and 17 are much easier to use)

**Course content**

1. C++11, some C++14 + some STL
2. Subset of C++ most useful for scientific computation – data structures, numerics, data processing

Ref book -> A tour of C++ Bjarne Stroustrup Addison-Wesley, 2014

**Day 1 - 20/02/2019**

1. C++ is a typed language.
2. We do not write iostream.h (that’s for C).
3. Different header files will add functions to the std namespace (using namespace std is not recommended).
4. g++ -std=c++14 –Wall –Wextra –g –o hello.exe hello.cpp

* -g flag for debugging info has no impact on performance. It only results into a slightly larger exec. It’s the –O0 flag added in a debug mode (usually in an IDE) which will reduce performance.
* –Wall flag is to include all compiler warnings. –Wextra flag is to include even extra compiler warnings.
* “return 0” statement return’s 0 to the Linux shell which indicates success status.

1. List initialization is preferred
   * int x {10.5}; (better than int x = 10.5; because the former will result into a warning while the latter will silently type cast float to int).
2. –lm flag is used to link against the libm.so library file to include the definition of math functions. It may so happen that –lm flag is not needed with some compilers which may link against libm.so by default. To disable default linking use –nodefaultlibs flag.
3. The gcc version 6 compiler supports flags for reporting indentation warnings.
4. Using arrays in C++ is not such a good idea as the programmer needs to keep track of the number of array elements separately.
5. Class methods can be classified into three types: constructors, inspectors, mutators, destructor.
6. Do not use std::pow to compute squares as it is 10X more expensive.
7. –O2 optimization level of g++ is safe (intel is a bit aggressive for –O2).

**Day 2 - 21/02/2019**

1. Makefile

CXX = g++

CXXFLAGS = -std=c++14 –g –O2 –Wall –Wextra

all: stats.exe

stats.exe: stats.o stats\_main.o

$(CXX) $(CXXFLAGS) –o $@ $^

stats\_main.o: stats\_main.cpp stats.h

$(CXX) $(CXXFLAGS) –c –o $@ $<

stats.o: stats.cpp stats.h

$(CXX) $(CXXFLAGS) –c –o $@ $<

clean:

$(RM) $(wildcard \*.o) $(wildcard \*.exe)

Difference between = and := ?

Writing your own make files is not a good idea. Better use CMake.

1. Check zeal online documentation browser.
2. Include guard in a header file is used to prevent unnecessary includes.
3. A function definition that cannot be in-lined should not be in a header file (recommendation).
4. A function definition in a .cpp file can be explicitly in-lined using inline keyword.
5. Do not overuse C macros.
6. Exception handling (check slides).
7. Containers
8. C++ containers are implemented in the STL (examples are array, valarray, vector, tuple, list, set, and map).
9. array is the simplest and was introduced in C++11. std::array container has some advantages over a C-style array (int a[]). Its interface makes it more convenient to find the size, and use with STL-style iterator-based algorithms. At the same time it is equally fast. std::array container allocates memory on the stack. <https://stackoverflow.com/questions/30263303/stdarray-vs-array-performance>
10. valarray is the next simplest and was introduced in C++03. It is variable size unlike array and has mathematical functions defined for numeric computation such as +, -, cos, sine etc. It also allows data slicing. Of course std::valarray container allocates memory on the heap. <https://stackoverflow.com/questions/8965562/what-is-the-difference-between-stdvalarray-and-stdarray>
11. vectors are sequence containers and even more convenient than an array or a valarray. The performance of vectors need to be understood. <https://stackoverflow.com/questions/1602451/c-valarray-vs-vector>
12. array, valarray and vector are contiguous in memory and therefore preferred for scientific computing. Tuple, list, set and map are not memory contiguous (needs a check).
13. The algorithms library defines functions for a variety of purposes (e.g. searching, sorting, counting, manipulating) that operate on ranges of elements. Algorithms will support parallel exec with gcc 9. Its already available with visual studio and will be available with intel compiler next year. Under the hood it will use OpenMP or TBB.

**Day 3 - 27/02/2019**

1. Advantages of C++ style cast static\_cast<double>(N) over C style (double)N cast:
   1. C++ style casts are checked by the compiler. C style casts aren't and can fail at runtime.
   2. C++ style casts can be searched for easily, whereas it's really hard to search for C style casts.
2. Explicit keyword before constructor(with one argument) of a class is to avoid implicit conversion (which a compiler is allowed to make) during function calls with that class object as parameter. <https://stackoverflow.com/questions/121162/what-does-the-explicit-keyword-mean>
3. Always use size\_t datatype over other unsigned datatypes
   1. The cstddef.h, cstdlib.h etc header files define a datatype called size\_t which is used to represent the size of an object. Library functions that take sizes expect them to be of type size\_t, and the sizeof operator evaluates to size\_t.
   2. When writing C++ code, it’s good to use size\_t whenever we are dealing with array indexing and loop counting (ex for indexing C++ containers, such as std::string, std::vector). <https://prateekvjoshi.com/2015/01/03/why-do-we-need-size_t/>

<https://www.embedded.com/electronics-blogs/programming-pointers/4026076/Why-size-t-matters>

* 1. size\_t is big enough to hold the size of the largest range our system can handle. Programs that use other types, such as unsigned int, for array indexing may fail on, e.g. 64-bit systems when the index exceeds [UINT\_MAX](https://en.cppreference.com/w/cpp/types/climits).
  2. size\_t can be unsigned int, unsigned long, or unsigned long long depending on the type. The compiler is free to choose the unsigned integer that’s big enough but not bigger than what’s needed. This results in performance gain.

1. Checking memory leaks at runtime
   1. The memory allocated by a std::vector is stored on the heap and its deallocation is taken care of by the compiler. Therefore, use std::vector over your own vector class. **Period.**
   2. std::vector<int> a; a.push\_back(10); Here a lives on the stack but value 10 lives on the heap.
   3. g++ -fsanitize=leak –g vector.cpp (to check memory leaks). -fsanitize option instruments the code during compilation. While running the code it prompts errors in case there are memory leaks. However, it may not be able to detect all memory leaks. Use valgrind for an extensive report.
   4. valgrind –leak-check=full ./a.out (to detect all the mem leaks and more). Note: do not use –fsanitize while compiling when using valgrind otherwise the system will slow down.
   5. cppcheck –enable=all –inconclusive vector.cpp (another useful tool to check code quality including memory leaks).
2. Always use & (call by reference) while passing large data containers. A single character can make a lot of difference.
3. Smart pointers
   1. Do not use raw pointers as they require new and delete operators which are prone to memory leaks if not used carefully.
   2. #include <memory>
   3. std::unique\_ptr<double[]> data; data=std::make\_unique<double[]>(n);
   4. As soon as pointer goes out of scope the data is freed so no need of garbage collection using delete.
   5. data.get() will return the raw pointer double\* to the unique\_ptr.
   6. Best practice: Try using STL container and if a container you want is not present (ex matrix) then create your own class with members from STL. If that is not sufficient and pointers are mandatory then use smart pointers.
4. Copy
   1. Copy constructor -> Vector(const Vector& other) : n {other.n} {}
   2. Copy assignment -> Vector& operator=(const Vector& other) {n = other.n; retrun \*this;}
   3. Copy constructor and copy assignment are both complementary.
   4. If STL containers are used, the compiler will call the copy constructor of the container, which will do the right job. That is another advantage of using STL containers rather creating your own container class.
5. Move (check gadget example in /CPlusPlus/UserDefinedTypes/ObjectLifeCycle)
   1. Move constructor -> Vector(Vector&& other) : n {other.n} {}
   2. Move assignment ->
   3. Looks similar to the copy constructor but moves instead of copying.
6. Vectors
   1. By default the size of a vector is zero in the beginning.
   2. Vectors have push\_back and insert methods. Since Vectors have contiguous data allocation, a push\_back is going to reallocate memory. To avoid memory reallocation use reserved.
   3. Complementary to push\_back and insert methods are emplace\_back and emplace. emplace\_back is better than push\_back as it creates Vector element in place (of course if there is a next free space available as vectors are contiguous).
   4. push\_back = move without copying
   5. emplace\_back = insert at the right place directly without even moving.
   6. Emaplce\_back is better than push\_back if there is contiguous memory available, otherwise it will involve copy constructor and will be terribly slow. Therefore be carfeul before using emaplace\_back. Use it wisely and preferably with a reserve keyword.

**Day 4 - 28/02/2019**

1. Templates
   1. Compiler explorer <https://godbolt.org/>
   2. Create a swap template and check its assembly expansion in the compiler explorer.
   3. STL as the name suggests is heavily templated.
   4. **Example**
   5. void printContainer(const std::string& s, const T& c) {cout s and range for over c;}
   6. printConatiner(“a”, a)
   7. In the above template, “a” is an rvalue (something which has no name in the calling context) and a is an lvalue (something which has name in the calling context).
   8. printConatiner(“a”, a + a) will give an error because a + a is not a valarray and therefore range for will fail. Also note a + a is an rvalue.
   9. Why a + a is not a valarray? Because the implementers of STL decided to keep it like this for efficiency. Think how many copy constructors would be needed to evaluate this expression a + sqrt(sqrt(a) + a^3 ), if the implementers would have decided to keep it otherwise.
   10. **Above example proves that care must be taken while using templates on containers.**
2. Typedef
   1. Forget typedef and start using using in C++11 !
3. Lambda functions
   1. Duck typing? (is related to templates where we can pass whatever and get things done.)
4. Inheritance
   1. private class members are only available to the base class and not to its derived class. Use protected instead of private to allow access by the derived class.
   2. MedianStats\* mstats = dynamic\_cast<MedianStats\*>(stats.get())
   3. Virtual member functions of a base class are run-time binded. That means they can be overridden by derived class members. Example virtual void PerformGlobalEval\_FRPM(); is a virtual member function and virtual void PerformGlobalEval\_FRPM()=0; is a pure virtual member function.
   4. Override specifier?
5. Friend functions
6. Floating point formatting
   1. Formats: scientific, fixed, defaultfloat (check slides)
   2. cout.width, cout.fill etc can also be useful.
   3. Ifstream, ofstream in #include <fstream>
   4. std::endl also flushes apart from inserting a new line and will hit performance big time. Use \n instead.
7. Numerics
   1. #include <complex> to deal with complex numbers (allows single/double precision).
   2. std::numeric\_limits<double>::min(), max(), epsilon()
   3. long double is extended precision (18 digits instead of 15 for double precision). Modern CPU’s have no hardware support for it and performance goes down the drain. If long double is absolutely necessary for your algorithm then rethink your algorithm.
   4. Random no generation (check slides). std:ref call for the engine is very important to avoid duplicate random numbers.
8. Linear algebra (check slides)
   1. Eigen (trivial to install since has only header files, but does not use BLAS and therefore not so performant).
   2. Armadillo (uses BLAS/Lapack, performant but no distributed algorithms).
9. Boost
   1. New versions are released very often.
   2. Some parts of Boost are not very stable.
10. Better to stick with C++11 or C++14 and not engage with C+17 unless latest compiler support is available.
11. Conclusions
    1. Use OpenMP/Threading Building Blocks (TBB) for multi-threading.
    2. TBB is task based rather than ….?
    3. TBB (according to Jan) is more performant then OpenMP.
    4. Advantage of TBB over OpenMP is clear when using containers such as lists which is not iterable using an unsigned index.
    5. Further reading (check slides).