ESS201: Programming-II
Module: C++

Jaya Sreevalsan Nair
jnair@iiitb.ac.in
International Institute of Information Technology, Bangalore

Term I: 2018-19; Lecture-01 on October 08, 2018

In 1970, Brian Kernighan and Dennis Ritchie designed a new language, famusly known as C, with a goal of writing operating systems. While C is a procedural language, it was lacking in object-oriented design (OOD). OOD is essential for grouping data and operations. Thus, in 1970, Bjarne Stroustrup designed "C with Classes", which was improved further and popularly came to be known as C++. Over time, C++ transformed itself from a "pre-compiler to C" to a language in its own right. It is the running in the top 4 programming languages, along with Java, C, and Python, in 2018, as per IEEE Spectrum https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018 .

1 C++ Programming Language Features

C++ is a high-level programming language [higher in comparison to C]. This means that the programmer is able to use an abstraction more in human-readable/natural language constructs than machine-readable one.

It is a compiled language, that is, a C++ program needs to be "written once, compiled anywhere" (WOCA). This is different from the philosophy of Java programming of "write once, run anywhere" (WORA). Thus C++ program upon compilation gives "object code", which is the machine language. A compiler is a software program that translates computer code/program from one language to another. Compilers translate from a high-level to a low-level language, while decompilers do the reverse. A C++ compiler, such as gcc, translates C++ code to object code. After compiling, several object codes are linked together along with a required set of libraries to create an executable program. The executable program is in machine language and is finally run. In order to run a C++ code, we do the following translations: `source` $\rightarrow$ `object` $\rightarrow$ `linked program` $\rightarrow$ `machine- executable`. Thus the starting point for a C++ programmer, just as any other high-level languages, is a text editor to write the source code.

Machine language is a language of numbers that represent instructions used by the computer; assembly language is where a single instruction translates directly to a single machine instruction. The difference between compiled and interpreted version of a language implementation is the use of a compiler and interpretor, in the first cut. The compiler "translates" multiple lines of code while the interpretor does the same, but for one line at a time. A compiler or an interpretor translates one statement to multiple assembly instructions.

The GCC (GNU C Compiler) is popularly used compiler for C++ programs. Initially, it was designed for C programs, and later extended to C++. Now GCC supports several front ends for Objective C, Objective-C++, etc. GCC is produced by the GNU (a recursive acronym for "GNU is Not Unix") project. In Unix, GCC provides different driver programs as an external interface. The driver program is gcc for C and g++ for C++. g++ upon compilation takes in a .cpp (or .cc or .C) file(s) and translates to object files with .o extension. GCC upon linking gives an executable, whose default name is a.out, unless specified by the programmer. While Windows uses .exe extension, Unix does not use any extension for executable

programs. GCC, as a compiler, is internally a compiler and a linker. Hence, compiling and linking can be done separately or together in terms of commandline instructions.

Memory management in C++ is a significant aspect of the language. C++ allows the programmer to perform dynamic memory allocation, using both stack and heap. Dynamic memory allocation implies that the memory is allocated at run time. We will see more of this in later lectures.

2 Example Codes and Practicals

We shall use several examples to articulate various aspects of C++ programming.

1. Installation of gcc (which includes g++) if not present in your system. For Ubuntu:
   ```
   $ sudo apt-get install build-essential
   ```

2. Understand the specifics of the compiler:
   ```
   $ man g++
   $ g++ -v
   ```

3. Demonstrating how to run a basic C++ program. Command-line instructions for compiling and running are:
   ```
   $ g++ hello.cpp ; ./a.out
   $ g++ -c hello.cpp ; ./a.out
   $ g++ -o output hello.cpp ; ./output
   ```

4. Use of namespace:
   ```
   $ g++ hello_variant.cpp ; ./a.out
   ```

5. Demonstrating compiling and linking; C++ vs C program:
   ```
   $ gcc mathtest.c ; ./a.out $ gcc mathtest.c -lm ; ./a.out
   $ g++ mathtest.cpp; ./a.out $ g++ -o output -lm mathtest.cpp ; ./output
   ```

6. Demonstrating comments, end-of-line comments in C++ vs block comments in C style:
   ```
   $ g++ comments.cpp; ./a.out
   ```

3 Application: Conway's Game of Life

Conway's Game of Life is the simplest two-dimensional cellular automaton, devised by John Horton Conway, in 1970. Cellular automaton, a discrete model studied in varied STEM (science, technology, engineering, mathematics), is a regular grid of cells, where each cell is in one of the finite number of states, e.g. on or off. The grid changes state based on a *fixed* rule applied to its neighborhood.

In Conway's life of game, each cell has eight neighbors (Moore neighborhood of range-1 cell, shown in Figure 1) and two states ("live" and "dead"). The rules applied in Conway's Game of Life allow transitions between live and dead states of the cell (i.e. between on and off states). At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if by underpopulation.

2. Any live cell with two or three live neighbors lives on to the next generation.

3. Any live cell with more than three live neighbors dies, as if by starvation or overpopulation.

4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

More on Conway's Game of Life at http://web.stanford.edu/~cdebs/GameOfLife/
Explanation of Conway's Game of Life at: https://www.youtube.com/watch?v=0XI6s-TGzSs

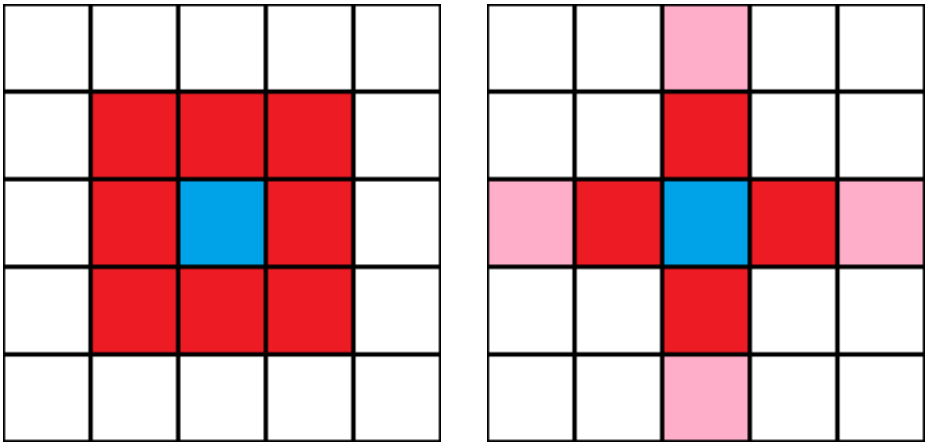

Figure 1: Different type of neighborhoods (in red) of each cell (in blue): (left) 8-cell neighborhood of Moore neighborhood, and (right) 4-cell neighborhood of von-Neumann neighborhood; if range is increased from 1 to 2, it becomes 8-cell neighborhood (red and pink cells).

References

- C++ Annotations, Practical C++
- https://en.wikipedia.org/wiki/Conway's_Game_of_Life
- http://mathworld.wolfram.com/CellularAutomaton.html