#7 Assignment - CMPT 383

Luiz Fernando Peres de Oliveira - 301288301 - lperesde@sfu.ca

I. Memory

Allocating memory in C++ (and most of languages) can be done statically or dynamically. In static allocated memory, all memory allocation calculations can be done before the program runs, while dynamic allocated memory can only be determined in runtime. Static memory is allocated by the system from the **stack** and dynamic memory is allocated by the system from the **heap**.

In C++, dynamic memory is usually allocated using the operator **new** or by using the C library *stdlib* directly. When you allocate dynamic memory in C++, the system returns a pointer to beginning of the sequence of bytes (block of memory). It is important to notice that memory is a finite resource and therefore, there are limitations on dynamic memory allocation. For this reason, a request to dynamic allocation of memory cannot be 100% guaranteed by the system.

Moreover, if an user tries to allocated more memory than it is physically available (the system can also limit a program's memory), it will be thrown an exception *bad_alloc*, informing that the system did not granted the requested memory. It is however possible to suppress such exception in C++ by using the keyword *nothrow*. Eg.: $mem = new(nothrow)int[3]$;

II. Paradigms

The lowest level abstraction of a computer programs is **machine code**, which is the binary assembly representation of a program (a machine code or *object code* contains literally 0's and 1's). Usually, high-level compilers (such as clang or clang++) compile their source code to Assembly language. From there, the Assembly code is then assembled to machine code by a program called Assembler. The Assembler assembles Assembly into machine code by mapping opcodes (Assembly instructions) in an usually one-to-one conversion, depending on the level of abstractions that the Assembly code may have and the Assembler may support.

Assembly Language (and therefore machine code) falls under the imperative paradigm because, Assembly, as any imperative language uses statements or instructions that change a program's state. In order to program in Assembly, one must describe all the steps the machine must take in order to accomplish a given task. In fact, high-level imperative programming languages such as *Fortran* and *C* are abstractions of Assembly Language.

Speaking of high-level and low-level programming languages, one might understand their main characteristics by their main differences. That is to say, a high-level language is a programming language (such as *Java* and *Python*) that allows programmers to write code that is not hardware specific, while a low-level programming is a very specific language that is possibly the closest mean of communication between a human and the bare metal, for instance Assembly and machine code. Moreover, high-level programming languages are more human-readable than low-level languages. Some say however that it exists another category between high-level and low-level so called mid-level programming languages, a category that puts *C*, *C++* and most system programming languages. This category would take a few characteristics of both low-level and high-level programming languages.

III. Functional Programming

IV. Programming Questions

1. Create an Object data structure in **C**.

```
typedef struct _Point {
  float x;
  float y;
} Point;
```

References

[1] "Function Types", Bartosz Milewski's Programming Cafe, 2017. [Online]. Available: https://bartoszmilewski.com/2015/03/13/function-types/. [Accessed: 29- Nov- 2017].

[2] "Dynamic memory - C++ Tutorials", Cplusplus.com, 2017. [Online]. Available: http://www.cplusplus.com/doc/tutorial/dynamic/. [Accessed: 04- Dec- 2017].

[3] A. code?, "Assembly code vs Machine code vs Object code?", Stackoverflow.com, 2017. [Online]. Available: https://stackoverflow.com/questions/466790/assembly-code-vs-machine-code-vs-object-code. [Accessed: 04- Dec- 2017].

[4] "Programming paradigm", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Programming_paradigm. [Accessed: 04- Dec- 2017].

[5] "Imperative programming", En.wikipedia.org, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Imperative_programming. [Accessed: 04- Dec- 2017].

[6] "Computers for Beginners/Programming - Wikibooks, open books for an open world", En.wikibooks.org, 2017. [Online]. Available: https://en.wikibooks.org/wiki/Computers_for_Beginners/Programming. [Accessed: 04- Dec- 2017].