Université Paris I – Panthéon-Sorbonne
90, rue de Tolbiac – 75634 Paris Cedex 13

Masters MAEF, QEM, DU MMEF — March 2016
Midterm exam

Introduction to Modern C++

Solutions

No documents, calculators, or computers allowed
Duration: 1h30.

Note: Please write clear and concise answers. Make sure that punctuation, if any, is visible. Most questions can be treated independently.

Question 1. C++ is a language designed for portability and expressiveness. The wide availability of free compilers of many platforms make it tool of choice over a variety of devices. The design of C++ keeps it close to low-level concerns and allows developers to create high-performance programs. Finally, C++ was influential to a generation of other programming languages.

Question 2. Compilation is, broadly, the process through which C++ code is translated into a format that a target device can directly execute, each device being different.

During this process, the compiler performs device-specific optimisations, adds library code and adjusts parameters in the program, going from C++ code to assembly code to object code, to finally output an executable program.

Without compilation, a program could not be directly run on the device.

Question 3. This is a shell command used to compile a C++ program. More precisely: `g++` is the name of the compiler (an alternative could be e.g. `clang++`), `another.cpp` is the file containing our C++ code, `something` is the output file that will contain the executable program, and `--std=c++11` signifies that we are using the 2011 revision of C++.

Should this command run without displaying errors, a new file `something` is created, that can be run by typing in a terminal `./something`.

Question 4. Type safety is a design principle, and best practice, that aim at guaranteeing the correct operation of programs. In C++, type safety is embodied through the requirement that variables are satically typed (types are assigned once and cannot be changed later) and that operations never mix different types. A violation of type safety may result in a compiler error or a runtime error.

Question 5. Describe the types associated with each variable:

1. Integer
2. Reference to an integer
3. Pointer to an integer
4. String of characters

5. Double precision floating-point numbers
6. Type inference: The compiler determines which type to assign `H`. Here probably `float`.
7. Vector: Resizeable array of double precision floating-point numbers
8. Array of arrays ("matrix") : Fixed-size array of 2 entries, each being an array with 2 integer entries.

Question 6. The type of `f(6)` is `double`, and its value is $6^3 = 216$. It is a pure function, since it has no side-effects : `f` behaves like a mathematical function.

Using `f` with large values may result in overflow, which would explain the observation.

Question 7. `std::vector<double>` or `std::vector<float>`

Question 8. Are these statements true or false, and why?

1. False. See Lab 2.
2. False. See Lab 2.
3. False. See Lab 2.
4. False. See Lab 2 or Qeustion 6.
5. False. It is easy to create a non-terminating loop, e.g. `for(;;){std::cout << "hi";}`.
6. False. Consequence of the above remarks and the fact that operation order is not the same.

```
#include <iostream>

int main() {
  for(int i = 0; i < 10; ++i) {
    std::cout << i << std::endl;
  }
}
```

Question 10 .Answer : `4, 16, 42, 20, 9, 3,`

Question 11. `#include <iostream>` is a pre-processor command indicating the requirement to prepend the `iostream` library to our program. The `iostream` library provides input and output functionality, e.g. `std::cout`. Other compiler directives include `#define` (macro definition) and `#pragma once` (include once).

Question 12. This is a recursive function. Recursion enables elegant and short programs that closely mimic mathematical notation. However, unless the compiler is designed to deal efficiently with recursion, execution of recursive programs may cause extensive memory usage and fail, sometimes silently.

`myFunction(n)` computes the sum of numbers from 1 to n, i.e. $(n+1)n/2$.

Question 12. The program prints

```
73
73
73
42
```

Indeed, `x, y` and `*z` are the same thing. The first two are bound because `y` is a reference to `x`, and `z` is the address of `y` so that `*z` is exactly `y`.

So when `y` is assigned the value 73 on line 5, `x, y` and `*z` all get the value 73.

But before that took place, `t` was assigned the value of `x` on line 4 — which was 42. Since `t` is not bound to any other variable, and is not modified after, it retains this value throughout the program.

`&y` is the address of `y`. `*z` is the content at address `z`.

Question 13. No. This is a consequence of Turing's theorem.

Indeed, assume that there exists some program T able to do this task, and let's show that this is a logical contradiction. T would take as input a program P, and return `true` if and only if P terminates. From T we can create the simple program T' that works as follows: on input P, if $T(P) =$ `true`, enter an infinite loop ; otherwise output `true`. The contradiction appears when considering $T'(T')$. Indeed, if T' terminates, then T' enters an infinite loop and thus doesn't terminate. If T' doesn't terminate, then it returns and thus terminates. In either case, we have a logical contradiction. As a result, T' doesn't exist, and thus T doesn't exist. QED.

Question 14. A field is a property of a class, it takes the form of a variable belonging to that class. A method is a responsibility of a class, and takes the form of a function belonging to that class.

By default, the visibility of class members is private.

```
class Duck {
  std::string name;
  int age;
  std::string position;

  public:
    void fly();
    void eat();
};
```

The visibility of members is to depend on the exact usage of this class. But as a general rule of thumb, we shall keep fields private — this is to avoid other parts of the program to alter the class internals, and possibly leave it in an incorrect state, which would violate encapsulation. The `fly` and `eat` methods could be made public.

Question 16. Through inheritance, children classes share the features and traits of their mother class. This is to be thought of as an "is-a" relationship: a duck is a bird, a bird is an animal.

Three main interests of inheritance are: (1) binding and overriding, i.e. the ability to deal with an object abstractly because it inherits from a class — e.g. Duck, Chicken and Ostrich are all kinds of Bird, so a function dealing with a Bird object could deal with either of those specific kinds of birds — yet retain its particular traits, e.g. all birds sing, but a duck quacks etc; and (2) code reuse : the child class retains all the public (and protected) fields and methods from the base class.

```
class Duck: public Bird {
  // As in question 15.
}

class Bird: public Animal {
  public:
    void eat();
    void fly();
}

class Animal {
  public:
    void eat();
}
```

Question 17. A header file is a design description file for C++ programs, that contains the definition of functions and objects to be used in a program. This is done for several reasons, chiefly amongst them : (1) separate design from code, (2) get rid of the requirement that function are defined before their calls, (3) expose an interface for other programs to use (e.g. when writing a library).

A header file typically has the `.h` extension, and is leveraged inside a C++ code file by the preprocessor command `#include "headerfile.h"`.

Question 18.

1. Line 1: Extra space after `iostream`
2. Line 5: Missing semi-color after `norm2()`
3. Line 9: Incorrect norm computation, should be `x*x+y*y`
4. Line 12: Incorrect return type for `main`, should be `int`
5. Line 13: Unused variables `y, z`.
6. Line 15: Should be `std::endl` instead of `endl`
7. Line 17: Should be `>>` instead of `<<`
8. Line 18: Extra space after `std`
9. Line 18: `myFunction` has not been declared
10. Line 18: Should be `std::endl` instead of `endl`
11. Line 21: Incorrect return type (type safety issue), should be `double`
12. Line 22: Should be `(x == 0)`
13. Line 23: Missing `<<` before `std::endl`
14. Line 23: Missing `return` statement in this branch, undefined return value!
15. Line 26: Type safety issue (on top of the return type being incorrect as pointed above), should be `42.0` instead of `42.`

Question 19. Line 14 is outside the scope of the preceding `if` statement and will always be executed, bypassing further tests. This mistake would have easily been avoided (or at least, spotted) if the programmer took care to use braces instead of dubious indentation.

```
#inclued <iostream>
#include <vector>
#include <algorithm>

int main() {
  std::vector<int> V {3, 1, 4, 1, 5, 9, 2, 6};

  std::cout << V[5] << std::endl;
  std::sort(V.begin(), V.end());
  std::cout << V[5] << std::endl;
}
```