MATH10017 Revision

Matteo Fasiolo

Some advice on the exam

- If you can not write proper code, try to write pseudo-code (rather than leaving a question blank).
- You can add comments to your answer, for instance if you forget how to call the modulus operator you can write

  ```
  if((j+pattern_type) MODULUS 2){ // I forgot the symbol for modulus
  ```

  Similar if you forgot about stdio.h you can write

  ```
  #include <STANDARD_C_LIBRARY_INPUT_OUTPUT.h> // I forgot what I need for printf
  ```

  and so on.
- The exam assumes that you know how to (not exhaustive list):
  1. write control-flow statements (e.g., for and while loops, if-else statements) in both C/C++ and R;
  2. define and use variables and functions in both C/C++ and R;
  3. access objects via indexes and or names (e.g. C arrays, R vectors, R lists, matrices, ...);
  4. write a basic C++ class with public and/or private members and methods (plus constructors and destructors);
  5. use basic functions and operators such as `printf`, `malloc` and % in C, or `sum`, `t` and `plot` in R;

Part A: Examples of Theoretical Questions

Questions and answers on C/C++:

1. *What are the advantages and disadvantages of using high- and low-level programming languages?* Low level:
   - (+) gives coder total control of hardware (see Rollercoaster Tycoon).
   - (+) can be efficient since it needs no translation (You talk to the computer using its native language!).
   - (-) can damage the hardware if the coder is not careful.
   - (-) is difficult to learn and read (machine instructions are usually very different from human languages).
   - (-) only works on a specific cpu architecture (e.g. x86 or ARM). Thus the code is not "portable".

   High level:
   - (+) Close to human language, easy to learn/read.
   - (+) CPU architecture independent, a.k.a., "Portable".
   - (-) Less efficient as the code requires "translation" before it can be executed on CPU.
   - (-) Cannot directly communicate with hardware: the coder has to interface with the abstraction.
2. *What does is mean to "compile" a program?* It means translating a program from a high level language (readable by a human) to a low-level language that can be directly processed by the CPU.
3. *What is the difference between "declaring" and "initialising" a variable in C.* A variable in C is a placeholder of some value. The value held by a variable can be changed later. In C, all variables must be declared, the syntax of declaration is `data_type variable_name` (e.g., int x;). Initialising means assigning a value to a variable that has already been declared (e.g., x = 1;). A variable can be declared and initialised on the same line (e.g., int x = 2).
4. *What is a local variable in C/C++?* A local variable is a variable that has been declared withing a function. It can only be accessed by the function where it is defined (unless it is returned) and not by other functions.
5. *Does stack memory within a C/C++ need to be released manually?* No, stack memory allocation and release are all automatically handled by the OS.

6. *What is the "virtual memory" of a computer?* Virtual Memory is an abstraction of the physical memory, made available to your program by the OS.

7. *What is a pointer in C/C++* A pointer is a variable that stores the address of another variable in the (virtual) memory.

8. *Does dynamic memory allocation happen on the stack or on heap memory*. On heap memory.

9. *What is the relation beween C and C++* C++ is a superset of C, which contains all language features in C and additional features (e.g., classes and other OOP features). Hence, a C program is also a valid C++ program but not vice-versa.

10. *You are given the files prog1.c and prog2.cpp provide the code for compiling them on the command line*. `gcc prog1.c -o prog1.out` and `g++ prog2.c -o prog2.out`.

11. *What is the purpose of the "constructor" and "destructor" in a C++ class?* The constructor is a class' method that specifies how specific instances of a class should be initiated. The destructor is also a class methods, that get called when a instance of the class goes out of scope, and that is used (e.g.) to de-allocate any dynamic memory allocated within the class.

12. *Describe the basic iterative steps of a local search algorithm* The steps are:

    (a) Local search algorithm starts from a candidate solution a problem.

    (b) It searches for a better solution that is close to the current solution.

    (c) If it find a better solution, it searches for an even better solution in its neighbourhood.

    (d) Keep iterating until a certain stopping criterion is satisfied. For local algorithms to work, the local problem must provide enough information to the original problem.

13. *Both greedy and local search algorithms are guaranteed to reach the globally optimal solution to a search problem. Do you agree with the previous sentence?* No, in general neither greedy or local algorithms are guaranteed to find the global solution. In fact, they rely on solving a subproblem at each iteration, which might lead them to reach a solution that is only a local optimum.

14. *What is the difference between a compiled and an interpreted language?* Code written in a compiled language must be translated to machine code (that runs on the CPU) by program called the compiler, prior to running. Code written in an intepreted scripts does not need compilation, but is run line by line by a program called the interpreter.

15. *Explain how vectorising R code can lead to faster computation.* Vectorising R code can lead to speed ups due to a) that fact that explicit R loops in R are avoided and b) vectorised code might be calling an highly optimised algorithm (often written in C/C++).

16. *What data structure(s) in R can be used to store object of different types (e.g., strings and numeric vectors)?* Lists.

Part A: Short programming questions

- Question A3 (2022 exam)

  1. What is stack memory and heap memory used for?
     Answer: Stack memory stores the function data and the current program being executed by the CPU. Heap memory stores data which are dynamically allocated during the runtime.
  2. Read the following program and answer questions.

```
#include <stdio.h>
void g(){
    printf("g is being called!\n");
}
void f(){
    g();
    printf("g has been called!\n");
}
void main(){
    f();
}
```

(a) Write down the messages printed on the screen after the above program is executed.
Answer:
g is being called!
g has been called!

(b) Draw the stack memory diagram when the program is printing out the following message:

```
g is being called!
```

Answer: Should draw three boxes: box labelled with g() on the top, f() in the middle and main() at the bottom.

(c) Draw the stack memory diagram when the program is printing out the following message:

```
g has been called!
```

Answer: Should draw two boxes: box labelled f() at the top and main() at the bottom.

- Question A4 (2022 exam). Read the following code and answer the questions.

```
#include <stdio.h>
class rectangle{
    int width;
    int height;
};

int main(){
    rectangle r1;
    r1.width = 5;
}
```

1. The above program will not compile. Add **one** C++ keyword in the code, so that it compiles.
   Answer: Add "public:" immediately after "class rectangle{".
2. However, by doing so, we have violated a key principle in OOP. Name this principle and discuss why such a violation can be dangerous.
   Answer: encapsulation. Exposing fields as “public” variables is dangerous. Someone can set the width to an arbitary value (e.g. -9999.)
3. This problem can be fixed by adding two methods in the rectangle class. Write down these methods. Answer:

```
#include <stdio.h>
class rectangle{
    int width;
    int height;
    public:
    int get_width(){
     return width;
    }
    void set_width(int w){
     width = w;
    }
};
```

- Question A5 (2022 exam). Read the following R program and answer the questions

```
n <- 5000
for (i in 1:n){
    is_square <- F
    for (j in 1:i){
        ---
    }
    if (is_square == T){
```

```
        print(i)
    }
}
```

1. Fill out the blank ("___") with no more than 4 lines of code so that the above program prints out all square numbers which are smaller than 5000. Square numbers are integers which can be written as the square of another integer.
   Answer:

   ```
   if(j*j == i){
       is_square <- T
   }
   ```

2. If we write the above algorithm in C code, would our program get slower or faster? Why?
   Answer: yes. The code above includes a double for loop, most of the computational comes from the inner loop (the one over j). As each iteration over j takes very little time, the overhead of interpretation can be significant.
3. If the above program takes 2 seconds to run, how long would it take when we set n to be 10000 (approximately speaking)? Why?
   Answer: approximately 8 seconds. The computational complexity of this algorithm is $O(n^2)$ or, more precisely, $O(1+2+3+...n) = O\left(\frac{n(n+1)}{2}\right)$.
4. Write a program which does exactly the same thing, but has a lower computational complexity.
   Answer:

   ```
   n <- 1
   while(n*n <= 5000){
     print(n*n)
     n <- n + 1
   }
   ```

Part B: Longer questions

- Question B2 (2022 exam)
  Write a **complete** C program which produces the following display output:

  ```
  *
  ***
  *****
  *******
  *********
  ```

  Your code must contain for loops.

  ```
  #include <stdio.h>

  void main()
  {
      for(int i = 0; i < 10; i+=2){
          for(int j =0; j<=i; j++){
              printf("*");
          }
          printf("\n");
      }
  }
  ```

  Further questions:

  1. Modify the program so that it prints out the alternative patterns:

```
*
*#*
*#*#*
*#*#*#*
*#*#*#*#*
```

or

```
#
#*#
#*#*#
#*#*#*#
#*#*#*#*#
```

Depending on the value of the binary variable `pattern_type`.

Answer:

```
  #include <stdio.h>

void main()
{
  int pattern_type = 0; // or 1
  for(int i = 0; i < 10; i+=2){
    for(int j =0; j<=i; j++){
      if((j+pattern_type) % 2){
        printf("#");
      } else {
        printf("*");
      }
    }
    printf("\n");
  }
}
```

2. Write the complete code for printing out sequences such as:

```
1 2

or

1 2 3 4

or

5 6 7 8 9 10
```

That is increasing sequences of neighboring integers, starting with `n1` and ending with `n2`. Further, prior to printing, the sequences must be stored in a (dynamically allocated) C array.

Answer:

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  int n1 = 4;
  int n2 = 15;
```

```
  int l = n2 - n1 + 1;
  int *v = malloc(l * sizeof(int));

  for(int ii = n1; ii <= n2; ii++){
    v[ii-n1] = ii;
  }

  for(int ii = 0; ii < l; ii++){
    printf("%d ", v[ii]);
  }

  printf("\n");

  // OR (alternative version)
  int *p = v;
  for(int ii = n1; ii <= n2; ii++){
    *p = ii;
    p++;
  }

  p = v;
  for(int ii = 0; ii < l; ii++){
    printf("%d ", *p);
    p++;
  }

  printf("\n");

  free(v);
}
```

- Question B2 (2022 Exam).

  Complete the following R program, so that it produces a matrix D, whose i,j-th entry $D_{i,j}$ is the taxicab distance between the i-th row of A and j-th row of B.

  Taxicab distance between two K-dimensional vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is defined as

$$d(\boldsymbol{a},\boldsymbol{b}) := \sum_{k=1}^{K} |a_k - b_k|,$$

  where a_k is the k-th element of $\boldsymbol{a}$ and $|a|$ is the absolute value of a. Your code should work for arbitrary $K > 0$.

```
A <- matrix(rnorm(100*2), nrow = 100)
B <- matrix(rnorm(100*2), nrow = 100)
D <- matrix(0, nrow = 100, ncol = 100)
# abs(a) computes the absolute value of a
# Your code starts here.
```

  Answer:

```
for(i in 1:nrow(A)){
    for(j in 1:nrow(B)){
        for(k in 1:ncol(A)){
            D[i,j] <- D[i,j] + abs(A[i,k] - B[j,k])
        }
    }
}
```

Extra questions:

- Provide a vectorised version of your solution above that uses no more that 2 for loops (unless you have already done it).
  Answer:

```
for(i in 1:nrow(A)){
    for(j in 1:nrow(B)){
            D[i,j] <- sum(abs(A[i, ] - B[j,]))
    }
}
```

- Provide a vectorised version of your solution above that uses no more that 1 for loop (unless you have already done it). Hint, consider the following code:

```
X <- matrix(1, 3, 3)
v <- 1:3
X - v
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]   -1   -1   -1
[3,]   -2   -2   -2
```

  and recall that `rowSums` and `colSums` compute the sum of the elements of a matrix across its rows or columns.
  Answer:

```
for(i in 1:nrow(A)){
    B_m_A <- t( t(B) - A[i, ] )
    D[i, ] <- rowSums( abs( B_m_A ) )
}
```