Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

1. Program behaviour, regardless of which evaluation strategy is used, should be identical, even though the order in which code executes is different. **T** [✓ **F**]

2. Pure virtual functions are not necessarily pure functions. [✓ **T**] **F**

3. The `filter` function in Python is an example of lazy evaluation. [✓ **T**] **F**

4. Python lambda function does not support multiple statements. [✓ **T**] **F**

5. A constexpr function or variable is exclusively for compile-time use. **T** [✓ **F**]

6. In C++, type inference for variable declaration (using the `auto` keyword) cannot fail [✓ **T**] **F**

Question 2. Multiple Choices

Which of the following operations are allowed inside a pure function?

[✓ i.] Read from a constant global variable.

ii. Read from a static function variable.

[✓ iii.] Modify a local variable.

[✓ iv.] Call another pure function.

v. Read from user input (e.g. using `std::cin`).

Question 3. Short Question

a) Describe three different optimization that can be made on code that is written in a referentially transparent style.

Common subexpression elimination: Since a pure function always returns the same value given the same argument, the compiler can store the common subexpression in a local temporary value instead of calculating it more than once. E.g.

`a = foo(1, 2)`
`b = foo(1, 2) + 5`
	➔	`temp = foo(1, 2)`
		`a = temp`
		`b = temp + 5`

Memoisation: With the same reasoning, we can cache values we have already calculated and return the cached value for subsequent invocation. E.g.

```
def foo(x, y):
    if (x, y) in foo.cache:
        return foo.cache[x, y]
    else:
        #
        # do expensive computation here
        #

        foo.cache[x, y] = result   # cache result of computation
        return result
foo.cache = dict()
```

Parallel computing: Since pure functions do not have any shared states, you can run independent parts of the code in parallel, e.g.

```
def bar(a, b):
    #
    # very expensive computation
    #
    return ret

# this is just pseudo-code, not valid Python
foo(
    thread(bar, (2, 8)),   # run in a different thread
    thread(bar, (5, 7))    # run in another thread
)
```

b) Given two Python lists of equal length, *a* and *b*, write an expression which evaluates to a list that contains the element-wise product and exclude all negative values. For example, suppose:

```
a = [2, -2, -3]
b = [4, -3, 1]
```

Then the returned list is `[8, 6]` (-3 was removed). Your solution may only use higher order functions and lambda functions.

```
list(filter(lambda x: x >= 0,
            map(lambda t: t[0] * t[1], zip(a, b))
))
```

Question 4. Programming Questions

Write a compile-time class, `ConstStr`, which provides the following three compile-time methods:

1. `hash`(), which returns a djb2 hash of the string (http://www.cse.yorku.ca/~oz/hash.html),
2. `startswith`(substr), which only returns true if the string starts with the substring substr, and
3. `endswith`(substr), which only returns true if the string ends with the substring substr.

`See conststr.cpp`