Additional considerations about functions

P. Ronchese
Dipartimento di Fisica e Astronomia “G.Galilei”

Università di Padova

“Object oriented programming and C++” course

Mathematical functions and program functions

Functions in a program are quite different than mathematical functions:

- mathematical functions give only one result depending on a well defined set of arguments, and nothing else;
- a function in a program may do many other operations:
  - change arguments when given by reference,
  - access (directly or indirectly) global variables or objects,
  - throw (directly or indirectly) exceptions,
  - ...

Side effects

A function in a program may have “side effects”:

- input-changing side effects
- pure side effects

Function arguments

The value or object returned by a function depends on its arguments (if any), but:

- `class` member functions do have an hidden argument, i.e. the pointer to object used to call the function (the `this` parameter),
- input-changing side effects can be excluded only when both the following conditions are satisfied:
  - the arguments are passed by value or reference-to-const,
  - a `class` member function is declared `const`;
- the result may depend on a global variable or object.

Repeated function calls

A function called more than once with the same arguments can return different results (e.g. `random()`).

Pure functions

A “pure function” is a function that:

- does return the same result when called with the same arguments,
- does not have any side effect.

Pure functions and code optimization

A pure function called in a loop with not-changing arguments can be moved outside the loop, to save computation time.

- Sometimes the compiler can recognize a pure function, and do itself such an optimization.
- Much more frequently, the compiler cannot determine if a function is pure or not, and cannot do that optimization.

Non-standard attributes - side effects

Some compilers allow to declare some attributes for functions, to inform the compiler itself they do not have side effect. These attributes are NOT-STANDARD, and consequently NOT-PORTABLE.

```
int main() {
  const Object o;
  const float x=5;
  int n=10;
  while(n--) cout << n+o.f(x) << endl;
  return 0;
}
```

If `float Object::f(float x)` does not have side effects it can be moved before the loop.

Non-standard attributes - access to global memory

Some compilers allow to declare some attributes for functions, to inform the compiler itself they do not access global memory. These attributes are NOT-STANDARD, and consequently NOT-PORTABLE.

```
int n;
int main() {
  const Object o;
  const float x=5;
  n=10;
  while(n--) cout << n+o.f(x) << endl;
  return 0;
}
```

If `float Object::f(float x)` does not access global memory it can be moved before the loop.