CoCoALib Programming

http://cocoa.dima.unige.it/

J. Abbott

Recap of C++

- C++ is evolving: C++03, C++11, C++14, C++17, C++20, ...
- small language + standard template library (STL)

C++ language

- explicit static typing (variables, fn return values)
- few basic types: `bool`, `int`, `double`, `char`
- `if ( cond ) { cmds } [ else { cmds } ]` ⟵ `then` is implicit!
- `for (int i = 1; i <= n; ++i) { cmds }`
- `for (const auto& x : v) { cmds }`
- `while ( cond ) { cmds }`
- `continue`, `break`, `return`
- user defined functions

C++ Standard Template Library

- contains many basic functions *e.g.* `sqrt`, `log`
- STL is very large (and evolving)
  - $\longrightarrow$ books by Scott Meyers
  - $\longrightarrow$ websites `cppreference`, `cplusplus`, ...
  - $\longrightarrow$ old book by Nikolai Josuttis
  - $\longrightarrow$ website `stackoverflow` (But be careful!)
- many "extended types"
  - `std::vector` $\longleftarrow$ array with indices from 0 to $n-1$ (not checked)
  - `std::string` for strings
  - input and output: `cin` $\longleftrightarrow$ keyboard `cout` $\longleftrightarrow$ screen
  - iterators
  - common algorithms, smart pointers, ...

Each Read-Only Parameter

- pass-by-value if data-structure is small ⟵ makes a copy
- pass-by-const-reference if data-structure may be big

Each Write-Only or Read-Write Parameter

- pass-by-reference (non-const)

```
bool IsPrime(long n);
bool IsPrime(const BigInt& N);
void QuoRem(long& q, long& r, long a, long b);
```

`void` ⟷ fn returns no value.

Recap of Object Oriented Programming

Object oriented is a set of guidelines for clean, safe programming.

An object is a value belonging to some "class" $\longleftrightarrow$ "type".

- an object comprises 0 or more (private) data members
- constructors create an object (from given initial arguments)
- destructor destroys an object, incl. related resources
- accessor functions ("setter" and "getter" fns)
- member fns, friend fns $\Longrightarrow$ direct access to data members
- (non-friend) non-member fns $\Longrightarrow$ no access to data members
- inheritance $\longleftrightarrow$ share common structure

Example: see `SmallPrime` (in `NumTheory-prime.H`), `ex-c++-class.C`

More advanced features of C++

- Exceptions:
  - alternative way of leaving a function — cascades “upwards”
  - typically used to “report errors”
  - need specific exception handlers
- class inheritance, virtual functions, “polymorphism”
- template classes and functions

These are used inside CoCoALib.
But you can use CoCoALib without knowing much about them.

Programming with CoCoALib

CoCoALib basic types:

- `BigInt` integers "without size limit"
- `BigRat` rationals "without size limit"
- `ring` various commutative rings CoCoA can represent
- `RingElem` element of a `ring` (often a polynomial or a coeff)

CoCoALib basic rings:

- `RingZZ()` ring of integers (created automatically)
- `RingQQ()` field of rationals (created automatically)
- `NewZZmod(p)` finite field (when `p` is prime)
- `NewPolyRing(RingQQ(), symbols("x,y,z"));`

CoCoALib documentation

- many example programs, names of the form `ex-XYZ.C`
- useful example: `ex-empty.C` ⟵ does (almost) nothing
- HTML manual pages
  ⟶ `/usr/local/include/CoCoA/doc/html/index.html`

Writing and Compiling

Easy way to start writing your program:

$\longrightarrow$ take a copy of `ex-empty.C` (or another example) and edit it.

Compilation: simpler via a `Makefile`

take a copy of `/usr/local/include/CoCoA/examples/Makefile`

If you do not want to copy `ex-empty.C` then the first thing to do is create `CoCoA::GlobalManager` object

$\longrightarrow$ constructor initializes CoCoALib "foundations"

$\longrightarrow$ destructor does final cleaning.

Exercises

- look at `ex-BigInt1.C`
- look at `ex-ring1.C`, `ex-RingQQ1.C`, `ex-RingElem1.C`
- look at many examples `ex-c++-XXX.C`, read them, understand.

Try compiling and running the examples *(after understanding them!)*

Exercise

Try completing `2021-10-L5-fibonacci.C` . . .
. . . and compiling it . . .
. . . and running it!

The End