MET CS 563
Software Development with C++ for Mathematical Finance

Dr. Tanya Zlateva
MET Computer Science Department
755 Commonwealth Ave., Room 103
Boston, MA 02215
phone: 617-353-2568
e-mail: “lastname-at-bu-dot-edu”

MET CS 563 Software Development with C++ for Mathematical Finance

Goals:

(i) provide knowledge and skills for designing and developing modular, scalable, maintainable programs in the C++ programming language using object-oriented methods;

(ii) discuss finite differences solutions for the basic models of financial derivatives; and

(iii) apply the knowledge in programming and numerical methods to design and develop software for modeling financial derivatives

Prerequisites:

- CAS MA 226 or equivalent knowledge of differential equations
- Prior programming experience in high level language recommended

NOTE: NO credit towards the MS in Computer Science

The science of computing – ancient algorithmic roots & latest technologies

Computer science is unique in that it

- Traces its intellectual roots as far back as the 3d millennium BC, when the scribes of ancient Mesopotamia recorded computing recipes - or *algorithms* as we will call them today - on clay tablets;
- Implements algorithms in the latest technologies, e.g. giga and tera flop supercomputers, optical computing.

Computing through the ages with "pencil and paper" - *software*

- Computing recipes in ancient Mesopotamia and Egypt on clay tablets (4th - 3d millennium BC);
- Euclid formulates the first formal algorithm that finds the greatest common devisor of two integers (ca. 300 BC);
- 9th century AD: Al-Khwarizmi writes textbooks on computing with

formulas → *algebra*

finite sequence of steps → *algorithm*

- Modern computing and software evolved through parallel and intersecting paths with contributions from mathematical logic, numerical methods, symbolic computation, language and automata theory, software engineering. For a quick overview and timeline see http://plato.stanford.edu/entries/computing-history/ and http://www.computer.org/cms/Computer.org/Publications/timeline.pdf

Computing through the ages with different "mechanical" devices - *hardware*

- *abacus*, invented some 5,000 years ago in Asia, performs digital computations (used in Asia through the 20th century);
- *mechanical calculators* in the 17th century: Wilhelm Schickard (Germany), Blaise Pascal (France), Gottfried Wilhelm Leibnitz (Germany);
- *analog devices*: slide rule, analog computers for special purpose computations.
- *person*, called *computer* till the mid 20th century, performing computations according to a given algorithm (sometimes several people each performing an given operation corresponding to the steps of the algorithm)

Blaise Pascal; Pascal's mechanical calculator; Wilhelm Schickard

Modern General Purpose Computers

The key difference of today's computers to their precursors is that they are

general purpose computing devices

i.e. they are not limited to solving a specific class of problems, such as arithmetic operations, differential equations, predicting tides, but can perform

any computation specified in a way the computer understands/accepts.

This generality led to the development of

- **a *new concept of computation*** and
- ***formal computer languages***

Historical Notes:

- 1938: Zuse built the first working general-purpose program-controlled digital computer in Germany,
- First fully functioning electronic digital computer was Colossus, used by the Bletchley Park (UK) cryptanalysts from February 1944.
- 1948 Neumann formulates the concept of a stored program
- 1951 Grace Murray Hopper builds first compiler at Harvard.

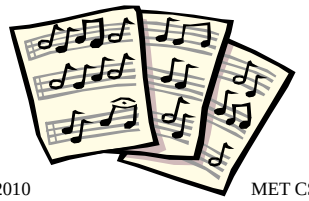
computation

*3.1415*2.1*2.1* — numerical

true and false = false — logical

$\int xdx = x^2/2$ — symbolic

"hello" ≠ "password" — text



any other data…, even the brain "computes"…

Computer Languages: high level vs. machine

Source code: program in a

high level language: independent of the specific computer architecture, e.g. Pascal, C, C++, Java

Compiler

Object code: translation of source code into another language, typically

machine language that is specific for a given architecture

Linker ← *Library Files* : additional code needed to run the program such as input/output , standard functions.

Executable code: additional code needed to run the program such as input/output , standard functions.

Computer Languages: imperative vs. object oriented

Imperative or *Procedural*: specify
- sequence of ***actions*** on
- ***data*** *(objects)*

do ***action****-1* on ***data****-object-1* *c=a+b*
do ***action****-2* on ***data****-object -1* *f=d*e*
…
do ***action-n*** on ***data****-object -n* *print c,f*

C

C++

Object Oriented: bundle actions & data together into ***objects,*** (e.g. date, place, person) with
- ***data*** or ***attributes****;* and
- ***behavior*** or actions object can do

Objects can
- interact with each other
- change each other,
- create new objects and

thus ultimately producing the desired result

object ***today***
attributes/data*:*
day 18
month 09
year 2008
behavior*:*
advance
tellMo/tellDay/tellYear

Type, Operations, Objects—Procedural

Formal specifications for:

(i) **type** and **operations** (actions) on data

`int` defines memory size for storing, interpreting, and modifying int objects

name
address *value/content*
int_size

\+ (int) *addition*

(ii) data **objects** with unique value of type size assigned in memory

a *&a* 5 *int_size*
b *&b* 10 *int_size*
c *&c* 15 *int_size*

(iii) applying operations to data

`c=a+b`

Type, Operations, Objects—Object Oriented

Formal specifications for:

(i) data are bundled with their operations

Date

attributes/data:

day
month
year

behavior:

advance()
tellMo()
tellDay()
tellYear()

(ii) defining objects

today

attributes/data:

day 19
month 01
year 2010

behavior:

advance()
tellMo()
tellDay()
tellYear()

(iii) applying actions on objects

```
today.advance()
today.tellMo()
today.tellDay()
today.tellYear()
```


Similarities and Differences of Type, Objects, Operations in Procedural vs. Object-Oriented

(i) type and operations

int

(ii) objects

a 5 *b* 10 *c* 15

(iii) operations on objects

```
c=a+b
```

Date

attributes/data:

day
month
year

behavior:

advance()
tellMo()
tellDay()
tellYear()

today

attributes/data:

day 8
month 9
year 2008

behavior:

advance()
tellMo()
tellDay()
tellYear()

```
today.advance()
today.tellMo()
today.tellDay()
today.tellYear()
```

Why C++?

- Remains the dominant programming language for large numeric software applications in engineering, finance, and science.
- Existence of large number of software packages written in in C and C++

Pros:

- Expressive
- Efficient
- Flexible—allows your own memory management
- Downward compatible with C
- Inherited C tool support.

Cons:

- Complex
- Difficult to learn
- Dangerous—allows your own memory management