

An Introduction to C++

Part 2

The Penna model
Introduction to classes

Segmenting programs

- ◆ Programs can be split into several files and linked together
 - ◆ However functions defined in another file have to be declared before use!
 - ◆ The function declaration is similar to the definition
 - ◆ but has no body!
 - ◆ parameters need not be given names
 - ◆ Easiest solution are header files. Help maintain consistency.
- ◆ file "RGP.h"

```
float square(float);
```
 - ◆ file "RGP.C"

```
#include "RGP.h"
float square(float x) {
    return x*x;
}
```
 - ◆ file "main.C"

```
#include <iostream>
#include "RGP.h"

int main() {
    std::cout << square(5);
}
```

Namespaces

- ◆ What if a `square` function is already defined elsewhere?
- ◆ **C-style solution:** give it a unique name; ugly and hard to type


```
float RGP_square(float);
```
- ◆ **Elegant C++ solution:** `namespaces`
 - ◆ Encapsulates all declarations in a modul, called “namespace”, identified by a prefix
 - ◆ Example:


```
namespace RGP
{
    float square(float);
}
```
 - ◆ Namespaces can be nested
- ◆ Can be accessed from outside as:
 - ◆ `RGP::square(5);`
 - ◆ `using RGP::square;`
`square(5);`
 - ◆ `using namespace RGP;`
`square(5);`
- ◆ Standard namespace is `std`
- ◆ For backward compatibility the standard headers ending in `.h` import `std` into the global namespace. E.g. the file “`iostream.h`” is:


```
#include <iostream>
using namespace std;
```

A first simulation: biological aging

- ◆ a simple model for death: $dN = -\lambda N dt$
- ◆ what about aging?
 - ◆ the remaining lifetime does not depend on current age.
 - ◆ true for radioactive decay
 - ◆ not true for biology
- ◆ what about age distribution?
 - ◆ exponential distribution!
 - ◆ also not what is seen in nature!
- ◆ what is missing?
 - ◆ some kind of aging
 - ◆ we need to develop a model containing aging

The Penna model

- ◆ A very simple model of biological aging
 - ◆ T.J.P. Penna, J. Stat. Phys **78**, 1629 (1995)
- ◆ Three important assumptions
 - ◆ finite age of adulthood
 - ◆ mutations of genetic material
 - ◆ limited resources
- ◆ This allows to model many features of biological population dynamics:
 - ◆ pacific salmon dies after giving birth
 - ◆ redwood trees have offsprings for hundreds of generations
 - ◆ catastrophic decline of cod in Atlantic due to small increase in fishing
- ◆ All these issues cannot be modeled without aging effects!

Details of the Penna model

- ◆ Each animal contains genes determining the survival rate
 - ◆ Each gene relevant for one year of its life
 - ◆ animal dies when it has collected T bad genes
- ◆ Limitation of resources
 - ◆ An animal that would survive because of its genes dies with a probability of N/N_0
 - ◆ N ...current population
 - ◆ N_0 ... maximum sustainable population
- ◆ Children
 - ◆ from an age of R years an animal gets a child asexually with a probability b (birthrate)
- ◆ Mutations
 - ◆ The children have the genes of the parents but with M random mutations

Program for the Penna model

- ◆ This our first exercise problem
- ◆ This week's tasks: find the concepts
- ◆ What are the abstract ideas?
 - ◆ genes
 - ◆ animal
 - ◆ Population
- ◆ Task
 - ◆ write a list of the properties of each of these entities
 - ◆ internal states
 - ◆ properties
 - ◆ operations
 - ◆ construction/destruction

Classes

- ◆ Are a method to create new data types
- ◆ Object oriented programming:
 - ◆ Instead of asking: "What are the subroutines?"
 - ◆ We ask:
 - ◆ What are the abstract concepts?
 - ◆ What are the properties of these concepts?
 - ◆ How can they be manipulated?
 - ◆ Then we implement these concepts as classes
- ◆ Advantages:
 - ◆ High level of abstraction possible
 - ◆ Hiding of representation dependent details
 - ◆ Presentation of an abstract and simple interface
 - ◆ Encapsulation of all operations on a type within a class
 - ◆ allows easier debugging

What are classes?

- ◆ Classes are collections of
 - ◆ functions
 - ◆ data
 - ◆ types
- ◆ representing one concept
- ◆ These “members” can be split into
 - ◆ `public`, accessible interface to the outside
 - ◆ should not be modified later!
 - ◆ `private`, hidden representation of the concept
 - ◆ can be changed without breaking any program using the class
 - ◆ this is called “data hiding”
- ◆ Objects of this type can be modified only by these member functions -> easier debugging

How to design classes

- ◆ ask yourself some questions
- ◆ what are the logical entities (**nouns**)?
 - ◆ -> classes
- ◆ what are the internal state variables ?
 - ◆ -> private data members
- ◆ how will it be created/initialized and destroyed?
 - ◆ -> constructor and destructor
- ◆ what are its properties (**adjectives**)?
 - ◆ -> public constant member functions
- ◆ how can it be manipulated (**verbs**)?
 - ◆ -> public operators and member functions