Functions

What are functions?

- They can be used to represent maths functions.
- More generally they are "reusable pieces of code".

In C++ one builds complex programs by composing functions.

Example

The function `blackScholesCallPrice` calls the functions

- `sqrt`,
- `normcdf`.

You will write these functions as homework exercises.

Example

The function `pricePortfolio` calls functions to:

- Read the trading position from a database.
- Read market data from Bloomberg.
- Price a vanilla European call.
- Price a knock-out American option.
- ...

Each of these functions calls more smaller functions.

- `blackScholesCallPrice` calls `sqrt` and `normcdf`.

The C++ function syntax

$$\texttt{compoundInterest} : \mathbb{R} \times \mathbb{R} \times \mathbb{Z} \longrightarrow \mathbb{R}$$

given by

$$\texttt{compoundInterest}(P, i, n) = P\left(1 + \frac{i}{100}\right)^n - P.$$

Corresponds to:

```
double compoundInterest(double P, double i, int n) {
    double interest = P * pow(1 + 0.01*i, n) - P;
    return interest;
}
```

Using a function

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

double compoundInterest(double P, double i, int n) {
    double interest = P * pow(1 + 0.01*i, n) - P;
    return interest;
}

int main() {
    int principal;
    double interestRate;
    int numberOfYears;
    cout << "How much are you investing ?\n";
    cin >> principal;
    cout << " What's the annual interest rate(%)?\n";
    cin >> interestRate;
    cout << "How long for ( years )?\n";
    cin >> numberOfYears;
    double interest
        = compoundInterest(principal,
        interestRate,
        numberOfYears);
    cout << "You will earn ";
    cout << interest;
    cout << "\n";
    return 0;
}
```

The structure of the file

Note the general syntax:

1) include statements
2) using statements
3) function A { . . . }
4) function B { . . . }

In our example there are two functions `main` and `compoundInterest`.
Functions are defined sequentially. You can't define a function inside another function.

What has this bought us?

- Reuse.
- Division of labour.
- Testability.
- Readability.

Type safety

Every variable, every parameter, every return type must have its type declared. The good:

- The compiler can find bugs.
- Code can be faster.
- Auto complete.

The bad:

- More typing.
- Discourages testing.
- You may become the compiler's slave.

Recursion

You can often define a sequence using recurrence relations.

$$x_n = nx_{n-1} \qquad n \geq 1$$

$$x_0 = 1$$

Clearly $x_n = n!$.

```
int factorial( int n ) {
    if (n==0) {
        return 1;
    }
    return n * factorial( n-1 );
}
```

Recursion

The good:

- Can make code easier to read.

The bad:

- Can make code harder to read.
- Can make inefficiencies hard to spot.
- Some languages are optimised for recursion. C++ is not.

The main method

- Specification for a command line program is that it must have a method called `main` that returns an integer.
- Specification for a windows application is that it must have a method called `WinMain` (that must take a specific set of parameters).
- Specification for a web app is ...
- Specification for an iPad app is ...

Libraries

- A library is a collection of functions that can be called by other applications and libraries.
- We will be writing a pricing library.
- We'll only use the `main` method for testing.

A challenging problem

You must declare a function before you can use it.

Example

- `blackScholesPrice` depends on `sqrt` and `normcdf`
- `pricePortfolio` depends on `readDatabase` and `blackScholesPrice`.

A valid order is `sqrt`, `normcdf`, `blackScholesPrice`, `readDatabase`, `pricePortfolio`.

Suppose we have 10000 functions and for each function we have a list of which other functions that depends upon. How would you choose a valid order of declarations?

Declaration and definition

This is why C++ allows you to separate declaration and definition.
A declaration:

```
double compoundInterest(double P, double i, int n);
```

A definition:

```
double compoundInterest(double P, double i, int n) {
    double interest = P * pow(1 + 0.01*i, n) - P;
    return interest;
}
```

Moving code around

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

double compoundInterest(double P, double i, int n);

int main() {
    int principal;
    double interestRate;
    int numberOfYears;
    cout << "How much are you investing ?\n";
    cin >> principal;
    cout << " What's the annual interest rate(%)?\n";
    cin >> interestRate;
    cout << "How long for ( years )?\n";
    cin >> numberOfYears;
    double interest
        = compoundInterest(principal,
        interestRate,
        numberOfYears);
    cout << "You will earn ";
    cout << interest;
    cout << "\n";
    return 0;
}

double compoundInterest(double P, double i, int n) {
    double interest = P * pow(1 + 0.01*i, n) - P;
    return interest;
}
```

Advice on code order

1. `include` statements.
2. `using` statements.
3. Declarations - most interesting first.
4. Definitions - most interesting first.

Declaration and definition of variables

Declaration:

```
double principal;
```

Definition:

```
principal = 1000.0;
```

Declaration and definition together:

```
double principal = 1000.0;
```

Functions that don't return a value

```
void printHello() {
    cout << "Hello\n";
}
```

Default values

```
double computePrice( double strike,
                     double timeToMaturity,
                     double spot,
                     double riskFreeRate,
                     double volatility,
                     double dividendRate = 0.0 );
```

Put the default value into the *declaration*.

Overloading

You can have two functions with the same name if they have different numbers of arguments.

```
double average( double a, double b ) {
    return 0.5 * (a+b);
}

double average( double a, double b, double c ) {
    return (a+b+c)/3.0;
}
```

Overloading

You can have two functions with the same name if they have different types of arguments.

```
int max( int a, int b ) {
    if (a>b)
        return a;
    return b;
}

double max( double a, double b ) {
    if (a>b)
        return a;
    return b;
}
```

Global variables

```
const double PI = 3.141592653589793;

double computeArea( int r ) {
    double answer = 0.5 * PI * r * r;
    return answer;
}

double computeCircumference( int r ) {
    double answer = 2.0 * PI * r;
    return answer;
}
```

Tip: Avoid using global variables

You should normally only use global variables to define constants. They make code very hard to read because they are GLOBAL. You need to understand ALL of the project to understand their effects. They violate information hiding.

Namespaces

- Is average really a good name for a function?
- There is a danger of name clashes.

```
#include <iostream>
// comment out the namespace
// using namespace std;
int main() {
    std::cout << "Hello World\n";
}
```