PDE Project Course

3. C++ programming

Karin Kraft and Johan Jansson

Department of Mathematical Sciences

Lecture plan

- History and overview
- Variables and basic operators
- The conditional statement: `if`
- Iteration: `for` and `while`
- Functions
- Classes
- Inheritance
- ...

- References
- Structuring your program
- Compiling a C++ program
- Introduction to DOLFIN
- Examples

History and overview

C++

- Invented by Bjarne Stroupstrup at AT&T Bell Laboratories in the early 1980's.
- Extends the C programming language to provide support for object-oriented programming.
- Widely used.
- Standardized by ANSI.

Hello world in C++

```
#include <iostream>

using namespace std;

int main()
{
    cout << ''Hello world!'' << endl;
    return 0;
}
```


Hello world in C++

```
#include <iostream>

using namespace std;

int main()
{
    int n = 10;
    for (int i = 0; i < n; i++)
        cout << ''Hello world!'' << endl;
    return 0;
}
```


A basic C++ vocabulary

- Fundamental data types:

  `int, float/double, bool, (char)`
- Conditions and loops:

  `if, else`
  `for, while`
- Classes:

  `class, public, private`
- General:

  `#include`

Comments

Use `//` and not `/* ... */` for comments:

```
// This is a comment
```


Variables and basic operators

Definition of a variable

- Note how in the Hello World-program each variable is defined as

```
VariableType nameOfVariable;
```

- A variable must be introduced before it is used.
- A variable can be defined almost anywhere in the code.

Fundamental data types

Type	Represents
`int`	Integers: ..., −2, −1, 0, 1, 2, ...
`float`	Real numbers: 0.1, 1.34e−15, −4.2e−6, ...
`bool`	Boolean value: `true` or `false`
`char`	Characters / bytes: 'a', 'F', '/', ...

`int` exists also in the versions `short` and `long`
`float` exists also in the version `double`

Examples

```
int n = 10;
int a = 1;
float x = 1.24;
double y = 3.14159265358979;
a = a + n;
y = 2.0 * y;
x = (1.25 / x) * 24.2;
int b;
int age = 20;
char c = 'a';
b = 17 / 2;
bool p = true;
```


Arithmetic operators

Operator	Represents
+	Addition
−	Subtraction
*	Multiplication
/	Division

- Note that there is no ˆ operator representing powers x^y!
- Shortcuts: +=, −=, *=, /=, ++, −−

Examples

```
int a = 2;
a += 3;
a *= 7;
a++;      //instead of a= a + 1;
a /= 5; //instead of a = a / 5;
a--;
a++;
a += 7;  //instead of a = a + 7;
```


The conditional statement:

`if`

The if-statement

The basic conditional statement in C++ is:

```
if ( <condition> )
{
    ...
}
else
{
    ...
}
```


Examples

```
if ( a > b || a < b )
    cout << ``a is not equal to b'' << endl;

if ( a > 3 && b > 3 )
    cout << ``a and b are greater than 3'' << endl;
elseif ( a > 3 )
    cout << ``a is greater than 3'' << endl;
elseif ( b > 3 )
    cout << ``b is greater than 3'' << endl;
else
    cout << ``neither a nor b greater than 3'' << endl;
```

Note also: ==, !=, >= and <=

Iteration: for and while

The for-statement

The basic iterative statement in C++ is:

```
for (<initialisation>; <condition>; <update>)
{

    ...

}
```

Examples

```
for (int i = 0; i < 10; i++)
    cout << ``i = `` << i << endl;

for (int j = 10; j > 0; j--)
    cout << ``j = `` << j << endl;
```


The `while`-statement

A somewhat different iterative statement in C++ is:

```
while (<condition>)
{

    ...

}
```


Examples

```
int i = 0;
while ( i < 10 )
{
    ...
    i++;
}

while ( true )
{
    ...
}
```


Functions

Declaration of a function

A function is a part of a program that can be called from other parts of the program.

```
<return type> <function name>(<variable>, ...)
{

    ...

}
```


Example

```
int max(int a, int b)
{
    if ( a > b )
        return a;
    else
        return b;
}
int main()
{
    int a = 3;
    int b = 5;
    int c = max(a, b);
    return 0;
}
```


Classes

Declaration of a class

```
class Vector {
public:
    Vector(int n);        // Constructor
    ˜Vector();            // Destructor

    void resize(int n); // A function
    void size( int & n );         // Another function
private:
    int n_;                // Size of the vector
    double_* values;       // The values
};
```

Private/Public

- Members of a class can be either *private* or *public*
- The members are by default supposed to be *private*
- *Public* members have to be declared *public* explicitly
- *Public* members can be used and reached from outside the class
- *Private* members can only be used by other members of the class class

Member data

The values stored in a class object is called *member data*.

Member data is usually declared *private*:

```
private:
    int n_;
    double* values_;
```


Member functions

Functions that belong to a class object are called *member functions*.

A class usually has a set of *public* member functions:

```
public:
    void resize(int n);
    int  size();
```


Constructors

The constructor function is called when an object of the class is created:

```
Vector x(10);
```

Now we have an object of the class *Vector* called x. We can call a member function with

```
x.resize( 20 )
```

Destructor

The destructor function is called when the object is destroyed, which happens when the object goes *out of scope*. This happens when we reach the end of the `{ }` block in which the object is defined.

User-defined operators

To use `Vector` objects as vectors, we implement the index operator `()`, the assignment operator `=`, and the arithmetic operators `+`, `−`, `*` and `/`.

```
class Vector {
public:
    ...
    double& operator()(int index);
    const Vector operator+(const Vector& vector);
    const Vector operator-(const Vector& vector);
    const Vector operator*(double a);
    const Vector operator/(double a);
    ...
};
```


Using the `Vector` class

```
Vector x(10);
Vector y(10);

for (int i = 0; i < 10; i++)
    x(i) = (double) i*i;

y = x + y;
x = x / 5.0 + y + x;
```


Inheritance

Parent class

```
class Figure {
public:
    Figure();      // Constructor
    ~Figure();     // Destructor

    virtual float area() = 0; // A function
    virtual float circumreference() = 0; // A function
};
```


Derived Class Rectangle

```
class Rectangle : public Figure() {
public:
    Rectangle( double b, double h ):Figure(),
    base_( b ), height_( h ){}
    ~Rectangle(){}
    float area(){
      return base_ * height_;
    }
    float circumreference(){
      return 2 * base_ + 2 * height_;
    }
private:
    double base_;
    double height_;
};
```


Derived Class Circle

```
class Circle : public Figure() {
public:
    Circle( double r ) : Figure(),
    radius_( r ) {}
    ~Circle(){}
    float area(){
      return 3.14 * radius_ * radius_;
    }
    float circumreference(){
      return 3.14 * 2 * radius_;
    }
private:
    double radius_;
};
```


References

References

A reference is an alias, i.e. an alternate name for some variable.

```
int a = 5;
int& b = a;

b += 7; // Now a and b equal 12!
```


References as function arguments

To allow a function to change a value, we can make the argument a reference:

```
void max(int a, int b, int& max_val)
{
    if ( a > b )
        max_val = a;
    else
        max_val = b;
}
```


References as function arguments

Another reason to use references can be to avoid unecessary copying of function arguments:

```
void max(Vector& x, double & maximum)
{
    maximum = x(0);
    for (int i = 1; i < x.size(); i++)
        if ( x(i) > maximum )
            maximum = x(i);
}
```


Continuation

```
void main() {
  Vector reallyHugeVector( 1000000 );
  double maximum;
  ...
  max( reallyHugeVector, maximum );
}
```

Structuring your program

Declaration and definition

A function does not have to be defined at the same time as it is declared:

```
int max(int a, int b);  // Somewhere...

int ClassName::max(int a, int b)   // and somewhere else
{
    if ( a > b )
        return a;
    else
        return b;
}
```


Header files and cpp-files

- Put all your declarations of a class in one file called the header file.
- Name the file ClassName.h
- Put the implementation of a class in a `cpp`-file.
- Name the file ClassName.cpp
- Use one file to one class.
- Include the corresponding header file.

Comment your code

- Write comments in your code
- Write short but good comments
- Your memory is not as good as you think!

Compiling a C++ program

Compiling Hello World I

```
elrond> g++ helloworld.cpp
elrond> ls
a.out  helloworld.cpp
elrond> ./a.out
Hello world!
```


Compiling Hello World II

```
elrond> g++ -o helloworld helloworld.cpp
elrond> ls
a.out  helloworld  helloworld.cpp
elrond> ./helloworld
Hello world!
```


Compiling Hello World III

```
elrond> g++ -c helloworld.cpp
elrond> ls
a.out  helloworld  helloworld.cpp  helloworld.o
elrond> g++ -o helloworld helloworld.o
elrond> ./helloworld
Hello world!
```

Writing a Makefile

```
CXX     = g++
CFLAGS  = -O2
LFLAGS  =
DEST    = helloworld
OBJECTS = helloworld.o
CXXLINK = $(CXX) -o $@

all: $(DEST)
clean:
        -rm -f *.o core *.core $(OBJECTS) $(DEST)
$(DEST): $(OBJECTS)
         $(CXXLINK) $(OBJECTS) $(CFLAGS) $(LFLAGS)
```


Compiling Hello World IV

```
elrond> make
g++     -c -o helloworld.o helloworld.cpp
g++ -o helloworld helloworld.o -O2
elrond> ./helloworld
Hello world!
elrond> make clean
rm -f *.o core *.core helloworld.o helloworld
```


Introduction to DOLFIN

Introduction to DOLFIN

- An adaptive finite element solver for PDEs and ODEs
- Developed at the Department of Computational Mathematics
- Written in C++
- Only a solver. No mesh generation. No visualization.
- Licensed under the GNU GPL
- `http://www.phi.chalmers.se/dolfin`

GNU and the GPL

- Makes the software free for all users
- Free to modify, change, copy, redistribute
- Derived work must also use the GPL license
- Enables sharing of code
- Simplifies distribution of the program
- Linux is distributed under the GPL license
- See `http://www.gnu.org`

Three levels

- Simple C/C++ interface for the *user* who just wants to solve an equation with specified geometry and boundary conditions.
- New algorithms are added at *module level* by the developer or advanced user.
- Core features are added at *kernel level*.

C++ examples in DOLFIN

- We will start looking at some of the basic infrastructure DOLFIN provides: linear algebra, input/output, mesh representation
- Demonstration

Examples