PDE Project Course

3. C++ programming

Anders Logg

logg@math.chalmers.se

Department of Computational Mathematics

Lecture plan

- History and overview
- Variables and basic operators
- `if`
- `for` and `while`
- Functions
- Classes
- Pointers and references
- Compiling a C++ program

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.
- Standardised by ANSI.
- Fundamental concept: *class*

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:

  `char, int, float, bool`

- Conditions and loops:

  `if, else, switch, case,`
  `for, while, break, continue`

- Classes:

  `class, public, private, protected`

- General:

  `#include, namespace, new, delete`

Comments

Use `//` or `/* ... */` for comments:

```
// This is a comment

/* This is also a comment, but this
   one stretches over several rows. */
```

Variables and basic operators

Definition of a variable

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

  `Type name;`

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

Fundamental data types

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

`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;
```

Derived types

Type	Represents
`*`	Pointer to
`&`	Reference to
`[]`	Array of

`*`, `&` and `[]` can be used together with the fundamental data types (or with user-defined classes) to create new types.

Examples

```
int a = 5;
int* b = &a; // Now *b is equal to 5
a = *b + a;  // Now  a is equal to 10

float x[10];
for (int i = 0; i < 10; i++)
    x[i] = (float) i; // Cast from int to float

int &c = a;  // Now c is equal to 10
a += 1;      // Increase a (and *b and c!) by one
```

More about pointers and references below!

Arithmetic operators

Operator	Represents
`+`	Addition
`–`	Subtraction
`*`	Multiplication
`/`	Division

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

Examples

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

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

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

`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;
}
```


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

Overloading of functions

Two functions can have the same name if they have different arguments:

```
int   max(int a, int b);

float max(float a, float b);
```


Classes

Declaration of a class

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

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


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 and destructors

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

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

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);
    Vector  operator+(Vector &vector);
    Vector  operator-(Vector &vector);
    Vector  operator*(double a);
    Vector  operator/(double a);
    ...
};
```

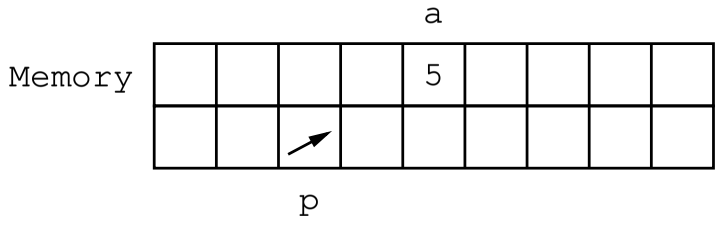

Using the `Vector` class

```
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;
```

Pointers and references

Pointers

A pointer contains the address of an object.



Using pointers

```
int a = 5;
int* p = &a;

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

Note the use of the *dereferencing* (indirection) operator `*` and the *address of* operator `&`. These should not be confused with `*` and `&` used to declare pointers and references!

Pointers as function arguments

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

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

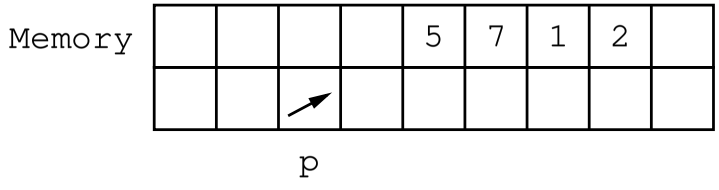
Pointers as function arguments

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

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


Allocating memory to a pointer

We can use a pointer to hold the address of a block of memory:



Allocating memory to a pointer

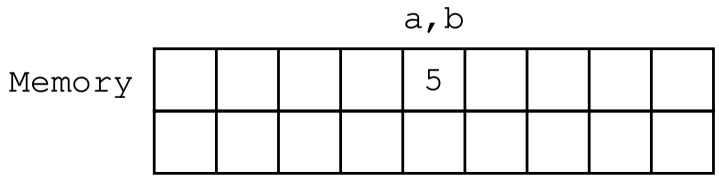
```
double*  a;
double*  x;
double** A;

a = new double;

x = new double[10];

A = new (double*)[10];
for (int i = 0; i < 10; i++)
    A[i] = new double[10];
```


Using `delete` to free memory

```
delete a;

delete [] x;

for (int i = 0; i < 10; i++)
    delete [] A[i];
delete [] A;
```

References

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



Example

```
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& c)
{
    if ( a > b )
        c = a;
    else
        c = b;
}
```


References as function arguments

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

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

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
```


Basic tools

- gcc / g++
- cpp
- make (gmake), Makefile
- automake, autoconf
- doc++
- gdb, ddd, valgrind
- xemacs, emacs, nedit, pico, vi
- tar, gzip
- top
- cvs

Compiling a GNU program

```
elrond> tar zxf dolfin.tar.gz
elrond> cd dolfin
elrond> ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets $(MAKE)... yes
checking for working aclocal-1.4... found
...
elrond> make
...
make[4]: Entering directory `/home/logg/work/src/dolfin/dolf
g++ -DPACKAGE_NAME=\"\" -DPACKAGE_TARNAME=\"\" -DPACKAGE_VER
g++ -DPACKAGE_NAME=\"\" -DPACKAGE_TARNAME=\"\" -DPACKAGE_VER
...
```