FEniCS Course

Lecture 17: C++ programming

Contributors
Anders Logg

What is C++

The C++ programming language is:

- General purpose
- Imperative
- Object-oriented
- Low-level (compared to Python)
- Fast
- Difficult

Computing $1 + 2 + \ldots + 100$ in Python

```
s = 0

for i in range(1, 101):
    s += i

print s
```

Running the program

Bash code

```
$ python sum.py
5050
```

Computing $1 + 2 + \ldots + 100$ in C++

C++ code

```
#include <iostream>

using namespace std;

int main()
{
  int s = 0;

  for (int i = 1; i < 101; i++)
    s += i;

  cout << s << endl;

  return 0;
}
```

Running the program

Bash code

```
$ g++ -o sum sum.cpp
$ ./sum
5050
```

Performance in Python vs C++

Let's instead compute $\sum_{k=1}^{N} k$ for $N = 100,000,000$.

Bash code

```
$ time python sum.py
5000000050000000

real 0m13.243s
```

Bash code

```
$ time ./sum
987459712

real 0m0.290s
```

Speedup by a factor 46 but strange results!

Modified C++ program

C++ code

```cpp
#include <iostream>

using namespace std;

int main()
{
  long int s = 0;
  long int N = 100000000;

  for (long int i = 1; i < (N + 1); i++)
    s += i;

  cout << s << endl;

  return 0;
}
```

New output

Bash code

```
$ time ./sum
5000000050000000

real 0m0.287s
```

C++ basics

Structure of a C++ program

C++ code

```
#include <stuff>

int main()
{
  // This is a comment

  code;
  code;

  return 0;
}
```

Declaring variables

C++ code

```
int a;
int b = 5;
float x = 3.5;
float y = 3;
double c;
c = 3.1415;
double d = 3.1415;
double e(3.1415);
bool f = true;
bool g = false;
auto h = g;
```

Useful types: `(unsigned) int`, `double`, `bool`

Illegal variable names

alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t, class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq

Comparison

C++ code

```
x == y;
x != y;
x > y;
x < y;
x >= y;
x <= y;
```

Logical operators

C++ code

```
!x;
x && y;
x || y;
```

If

C++ code

```
if (x > y)
   x += y;

if (x > y) x += y;

if (x > y) { x += y; }

if (x > y)
{
   x += y;
}
```

If / else

C++ code

```
if (x > y)
{
   x += y;
   y += x;
}
else if (x < y)
{
   x += 1;
}
else
{
   y += 1;
}
```

For loop

C++ code

```
for (<init>; <condition>; <update>)
  stuff;

for (int i = 0; i < 100; i++)
  stuff;

for (int i = 0; i < 100; i++)
{
  stuff;
  morestuff;
}
```

While loop

C++ code

```
while (<condition>)
  stuff;

int i = 0;
while (i < 100)
{
  stuff;
  i++;
}

int i = 0;
while (true)
{
  stuff;
  if (i == 99)
    break;
}
```

Functions

C++ code

```
<type0> myfunction(<type1> x, <type2> y, ...)
{
  <type0> z;

  return z;
}

double sum(double x, double y)
{
  double z = x + y;
  return z;
}

double sum(double x, double y)
{
  return x + y;
}
```

Pass by reference/value

C++ code

```
double sum(const double& x, const double& y)
{
  return x + y;
}

double sum(const double& x, double& y)
{
  y += 1; // bad practice!
  return x + y;
}
```

Pass built-in types by value.
Pass custom (heavy) types by reference.

Variables, references and pointers

A variable is a value associated with a name.
A reference is an alias for a variable.
A pointer is an address to a variable.

C++ code

```
double x = 3;
double& y = x;
double* z = &x;

y += 1;  // changes x and y (and *z)
*z += 1; // changes x and y (and *z)
z += 1;  // adds 8 bytes to address
*z += 1; // gives a segmentation fault (?)
```

See drawing on blackboard for an illustration!

C++ classes

Class structure

C++ code

```
class Foo
{
public:

  void foo()
  {
    stuff;
  }

  void bar()
  {
     bar;
  }

};

Foo f;
f.foo();
f.bar();
```

Class members

C++ code

```
class Foo
{
public:

  void foo() // this is a member function
  {
    stuff;
  }

  int x; // this is a member variable

};
```

Public and private class members

C++ code

```
class Foo
{
public:

  int x;

private:

  int y;

}

Foo f;
f.x += 1; // ok
f.y += 1; // not ok
```

Constructor / destructor

C++ code

```
class Foo
{
public:

  Foo(int x) : _x(x)
  {
    stuff;
  }

  ~Foo()
  {
    cleanup;
  }

private:

  int _x;

};
```

Operator overloading

C++ code

```
class Foo
{
public:

  Foo operator+(const Foo& y)
  {
    Foo z;
    stuff;
    return z;
  }

};

Foo x;
Foo y;
Foo z = x + y;
```

Overloadable operators

C++ code

```
+ - * / % ^
& | ~ ! , =
< > <= >= ++ --
<< >> == != && ||
+= -= /= %= ^= &=
|= *= <<= >>= [] ()
-> ->* new new [] delete delete []
```

Further reading

Important topics not covered in this lecture:

- Using `new` / `delete`
- Using C++ arrays: `double x[3];`
- Using the STL library: `std::vector` etc
- Using C++ templates: `template class<T> ...`
- Building and linking complex programs