Basic syntax in C++

Shahram Rahatlou

Computing Methods in Physics

`http://www.roma1.infn.it/people/rahatlou/cmp/`

Anno Accademico 2019/20

SAPIENZA
UNIVERSITÀ DI ROMA

Brief History of C

- C was developed in 1967 mainly as a language for writing operating systems and compilers
  - Think about the gcc compiler and Linux today
  - You can compile the gcc compiler yourself
  - You can get the latest linux kernel (core of the linux operating system) from www.kernels.org and compile it yourself

- C was the evolution of two previous languages: B and BCPL
  - Both used to develop early versions of UNIX at Bell Labs

- C became very popular and was ported to variety of different hardware platforms
  - C was standardized in 1990 by International Organization for Standardization (ISO) and American National Standards Institute (ANSI)
    - ANSI/ISO 9899: 1990

Object Oriented Programming and Birth of C++

- By 1970's the difficulties of maintaining very large software projects for companies and businesses had lead to structured programming
  - From Wikipedia:

    Structured programming can be seen as a subset or subdiscipline of procedural programming, one of the major programming paradigms. It is most famous for removing or reducing reliance on the GOTO statement (also known as "go to")

- By late 70's a new programming paradigm was becoming trendy: object orientation

- In early 1980's Bjarne Stroustrup developed C++ using features from C but adding capabilities for object orientation

What is 'Object Oriented Programming` anyway?

- Objects are software units modelled after entities in real life
  - Objects are entities with attributes: length, density, elasticity, thermal coefficient, color
  - Objects have a behavior and provide functionalities
    - A door can be opened
    - A car can be driven
    - A harmonic oscillator oscillates
    - A nucleus can decay
    - A planet moves in an orbit
- Object orientation means writing your program in terms of well defined units (called objects) which have attributes and offer functionalities
  - Program consists in interaction between objects using methods offered by each of them

C++ is not C !

- Don’t be fooled by the name!

- C++ was developed to overcome limitations of C and improve upon it
  - C++ looks like C but feels very differently
  - C++ shares many basic functionalities but improves upon many of them
    - For example input/output significantly better in C++ than in C

- C excellent language for structural programming
  - Focused around actions on data structures
  - Provides methods which act on data and create data

- C++ focused on inter-action between objects
  - Objects are ‘smart’ data structures: data with behavior!

What You Need to compile your C++ Program?

- On Linux machines you should have the g++ compiler installed by default
- On Windows you can use the C++ compiler provided by the free version of Visual Studio
- On Mac OS, you can install the g++ compiler via XCode, available for free on Mac App Store

- The easiest way is to use the virtual box available on the course website

Structure of a C++ Program

```
// your first C++ application!
#include <iostream> // required to perform C++ stream I/O

// function main begins program execution
int main() {
   return 0; // indicate that program ended successfully

} // end function main
```

Precompiler/Preprocessor Directives

What is the preprocessor? What does it do?

```
// your first C++ application!
#include <iostream> // required to perform C++ stream I/O

// function main begins program execution
int main() {
   return 0; // indicate that program ended successfully

} // end function main
```

`iostream` will be included before compiling this code!

What does the Preprocessor do?

```
// Foo.h
class Foo {
public:
 Foo() {};
 Foo(int a) { x_ = a; };

private:
 int x_;
};
```

```
// ExamplePreprocessor.cpp
#include "Foo.h"


int main() {

  return 0;
}
```

Pre-compile only

```
$ g++ -E ExamplePreprocessor.cpp > prep.cc
$ cat prep.cc
# 1 "ExamplePreprocessor.cpp"
# 1 "<built-in>"
# 1 "<command line>"
# 1 "ExamplePreprocessor.cpp"

# 1 "Foo.h" 1
class Foo {
public:
 Foo() {};
 Foo(int a) { x_ = a; };

private:
 int x_;
};
# 3 "ExamplePreprocessor.cpp" 2

int main() {

  return 0;
}
```

- Replace user directives with requested source code
  - Foo.h is included in ExamplePreprocessor.cpp

Comments in C++

```
// your first C++ application!
#include <iostream> // required to perform C++ stream I/O

// function main begins program execution
int main() {
   return 0; // indicate that program ended successfully

} // end function main
```

- Comments preceded by //
  - Can start anywhere in the program either at the beginning or right after a statement in the middle of the line

Compiling a C++ application

```
$ g++ -o Welcome Welcome.cpp
```

Name of the binary output

C++ file to compile and link

```
$ ls -l
-r--r--r--  1 rahatlou None   1379 Apr 18 22:55
Welcome.cpp
-rwxr-xr-x  1 rahatlou None 476600 Apr 18 22:57
Welcome
```

- We will be using the free compiler gcc throughout the examples in this course

Some basic aspects of C++

- All statements must end with a semi-colon ;
  - Carriage returns are not meaningful and ignored by the compiler

- Comments are preceded by //
  - Comments can be an entire line or in the middle of the line after a statement

- Any C++ application must have a **`main`** method

- **`main`** must return an **`int`**
  - Return value can be used by user/client/environment
  - E.g. to understand if there was an error condition

What about changing a different type of **`main`**?

```
// VoidMain.cpp
#include <iostream>
using namespace std;

void main(){

  // no return type

} // end function main
```

```
$ g++ -o VoidMain VoidMain.cpp
VoidMain.cpp:6: error: `main' must return `int'
```

- Compiler requires **`main`** to return an **`int`** value!
- Users must simply must satisfy this requirement
  - If you need a different type there is probably a mistake in your design!

Typical Compilation Errors So Far

```
// BadCode1.cpp
#include <iostream>
using namespace std;

int main() { // main begins here

   int nIterations;

   cout << "How many
           iterations? "; // cannot break in the middle of the string!

   cin >> nIteration; // wrong name! the s at the end missing

   // print message to STDOUT
   cout << "Number of requested iterations: " << nIterations << endl;

   return 0 // ; is missing!

} // end of main
```

```
$ g++ -o BadCode1 BadCode1.cpp
BadCode1.cpp: In function `int main()':
BadCode1.cpp:9: error: missing terminating " character
BadCode1.cpp:10: error: `iterations' undeclared (first use this function)
BadCode1.cpp:10: error: (Each undeclared identifier is reported only once for each function
 it appears in.)
BadCode1.cpp:10: error: missing terminating " character
BadCode1.cpp:12: error: `nIteration' undeclared (first use this function)
BadCode1.cpp:12: error: expected `:' before ';' token
BadCode1.cpp:12: error: expected primary-expression before ';' token
BadCode1.cpp:19: error: expected `;' before '}' token
```

Some C reminders

Always initialise your variables!

```
// tinput_bad2.cc
#include <iostream>
using namespace std;

int main() {


  int iters;
  cout << "iters before cin: " << iters << endl;

  cout << "iterations? ";
  cin >> iters;

  cout << "requested " << iters << " iterations" << endl;

  return 0;
}
```

Random value since not initialized!

```
$ g++ -Wall -o tinput_bad2 tinput_bad2.cc
$ ./tinput_bad2
iters before cin: 134514841
iterations? 3
requested 3 iterations
$ ./tinput_bad2
iters before cin: 134514841
iterations? er
requested 134514841 iterations
```

Arrays (same as in C)

```
// vect3.cc
#include <iostream>
using namespace std;

int main() {


   float vect[3] = {0.4,1.34,56.156}; // vector of int
   float v2[3];
   float v3[] = { 0.9, -0.1, -0.65}; // array of size 3

   for(int i = 0; i<3; ++i) {
     cout << "i: " << i << "\t"
          << "vect[" << i << "]: " << vect[i] << "    \t"
          << "v2[" << i << "]: " << v2[i] << " \t"
          << "v3[" << i << "]: " << v3[i]
           << endl;
   }

   return 0;
}
```

Index of arrays starts from 0 !!

v2[0] is the first elements of array **v2** of size 3.

v2[2] is the last element of **v2**

What happened to **v2**?

```
$ g++ -o vect3 vect3.cc
$ ./vect3
i: 0    vect[0]: 0.4          v2[0]: 5.34218e+36      v3[0]: 0.9
i: 1    vect[1]: 1.34         v2[1]: 2.62884e-42      v3[1]: -0.1
i: 2    vect[2]: 56.156       v2[2]: 3.30001e-39      v3[2]: -0.65
```

Arrays and Pointers

- The name of the array is a pointer to the first element of the array

```
// array.cpp
#include <iostream>
using namespace std;

int main() {

   int vect[3] = {1,2,3}; // vector of int
   int v2[3]; //what is the default value?
   int v3[] = { 1, 2, 3, 4, 5, 6, 7 }; // array of size 7

   int* d = v3;
   int* c = vect;
   int* e = v2;

   for(int i = 0; i<5; ++i) {
     cout << "i: " << i << ", d = " << d << ", *d: " << *d;
     ++d;
     cout << ", c = " << c << ", *c: " << *c;
     ++c;
     cout << ", e = " << e << ", *e: " << *e << endl;
     ++e;
   }
   return 0;
}
```

What happened to `e`?

```
$ g++ -o array array.cc
$ ./array
i: 0, d = 0x23eec0, *d: 1, c = 0x23eef0, *c: 1, e = 0x23eee0, *e: -1
i: 1, d = 0x23eec4, *d: 2, c = 0x23eef4, *c: 2, e = 0x23eee4, *e: 2088773120
i: 2, d = 0x23eec8, *d: 3, c = 0x23eef8, *c: 3, e = 0x23eee8, *e: 2088772930
i: 3, d = 0x23eecc, *d: 4, c = 0x23eefc, *c: 1627945305, e = 0x23eeec, *e: 2089866642
i: 4, d = 0x23eed0, *d: 5, c = 0x23ef00, *c: 1876, e = 0x23eef0, *e: 1
```

Another bad example of using arrays

```
// vect2.cc
#include <iostream>
using namespace std;

int main() {


   float vect[3] = {0.4,1.34,56.156}; // vector of int
   float v2[3]; // use default value 0 for each element
   float v3[] = { 0.9, -0.1, -0.65, 1.012, 2.23, -0.67, 2.22 }; // array of size 7

   for(int i = 0; i<5; ++i) {
     cout << "i: " << i << "\t"
          << "vect[" << i << "]: " << vect[i] << "  \t"
          << "v2[" << i << "]: " << v2[i] << "  \t"
          << "v3[" << i << "]: " << v3[i]
           << endl;
   }

   return 0;
}
```

Accessing out of range component!

```
$ g++ -o vect2 vect2.cc
$ ./vect2
i: 0    vect[0]: 0.4            v2[0]: 5.34218e+36      v3[0]: 0.9
i: 1    vect[1]: 1.34           v2[1]: 2.62884e-42      v3[1]: -0.1
i: 2    vect[2]: 56.156         v2[2]: 3.30001e-39      v3[2]: -0.65
i: 3    vect[3]: 5.60519e-45    v2[3]: 1.57344e+20      v3[3]: 1.012
i: 4    vect[4]: 1.72441e+20    v2[4]: 0.4              v3[4]: 2.23
```

Example of Bad non-initialized Arrays

```
// vect1.cc
#include <iostream>
#include <cmath>

using namespace std;


int main() {

   float vect[3]; // no initialization

   cout << "printing garbage since vector not initialized" << endl;
   for(int i=0; i<3; ++i) {
     cout << "vect[" << i << "] = " << vect[i]
          << endl;
   }

   vect[0] = 1.1;
   vect[1] = 20.132;
   vect[2] = 12.66;

   cout << "print vector after setting values" << endl;
   for(int i=0; i<3; ++i) {
     cout << "vect[" << i << "] =    " << vect[i] << "\t"
          << "sqrt( vect[" << i << "] ) = " << sqrt(vect[i])
          << endl;
   }

   return 0;
}
```

```
$ ./vect1
printing garbage since vector not initialized
vect[0] = 2.62884e-42
vect[1] = NaN
vect[2] = 0
print vector after setting values
vect[0] =    1.1        sqrt( vect[0] ) = 1.04881
vect[1] =    20.132     sqrt( vect[1] ) = 4.48687
vect[2] =    12.66      sqrt( vect[2] ) = 3.55809
```

Control Statements in C++

```
// SimpleIf.cpp
#include <iostream>
using namespace std;

int main() { // main begins here

   if( 1 == 0 ) cout << "1==0" << endl;

   if( 7.2 >= 6.9 ) cout << "7.2 >= 6.9" << endl;

   bool truth = (1 != 0);
   if(truth) cout << "1 != 0" << endl;

   if( ! ( 1.1 >= 1.2 ) ) cout << "1.1 < 1.2" << endl;

   return 0;
} // end of main
```

```
$ g++ -o SimpleIf SimpleIf.cpp
$ ./SimpleIf
7.2 >= 6.9
1 != 0
1.1 < 1.2
```

Declaration and Definition of Variables

```
// SimpleVars.cpp
#include <iostream>
using namespace std;

int main() {

   int samples; // declaration only

   int events = 0; // declaration and assignment

   samples = 123; // assignment

   cout << "How many samples? " ;
   cin >> samples; // assigment via I/O

    cout << "samples: " << samples
         << "\t"  // insert a tab in the printout
         <<  "events: " << events
         << endl;

   return 0;
} // end of main
```

```
$ g++ -o SimpleVars SimpleVars.cpp
$ ./SimpleVars
How many samples? 3
samples: 3      events: 0
```

Loops and iterations in C++

```
int main() { // main begins here

   int nIterations;
   cout << "How many iterations? ";
   cin >> nIterations;

   int step;
   cout << "step of iteration? " ;
   cin >> step;

   for(int index=0; index < nIterations; index+=step) {
     cout << "index: " << index << endl;
   }
   return 0;
} // end of main
```

Maximum

Starting value

step

```
$ g++ -o SimpleLoop SimpleLoop.cpp
$ ./SimpleLoop
How many iterations? 7
step of iteration? 3
index: 0
index: 3
index: 6
```