Modern C++ - Basics

1. Basics

The C++ programming language

C++ (/ˈsiː plʌs plʌs/) is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation.

(from wikipedia

History of C++

Origins

```
 Assembly -> BCPL -> B -> C          -> D
                              -> C++ -> Java
           Algol -> Algol68          -> C#
         Fortran -> Simula67
```

Timeline

- 1980 Bjarne Stroustrup starts working on *C with Classes*
- 1983 The new language is named as C++
- 1985 Aleksey Stepanov works on *Generic programming*
- 1990 The language has *exceptions*
- 1990 The Annotated Reference Manual (ARM) (book from Stroustrup and M. Ellis)
- 1991 ISO standardisation process starts
- 1994 The Design and Evolution of C++ (book from Stroustrup)
- 1998 C++ standard ISO/IEC 14882:1998
- 2003 Minor bugfixes (C++03 standard)
- 2011 Major language revision (C++11 standard)
- 2014 Minor revision (C++14)
- 2017 Major language revision (C++17)

Design goals of C++

Type safety

C++ is a statically, strongly typed programming language The compiler decides the type of all (sub)expression in compile time

In run time: pointers, conversions, Object-oriented constructs brings dynamism into the static type system.

Resource safety

- Not just memory! All resources (files, sockets, locks, etc.)
- No garbage collection by def. (but can implement)
- Use the RAII (Resource Acquisition Is Initialization) idiom

Many beginner makes resource errors.

Performance

- Direct access to HW resources. No virtual machine
- High performance trading, phone exchange systems
- Low energy cosumption (Mars rover)

C++ programs does not guarantees high performance! They gives control to the programmer to decide on performance-related issues.

Predictability

Orthogonal features should work well together. Capability to safety implement large systems (2-10 million eLoC).

Learnability, readability

Lots of problems in earlier versions. C++11 version goes from expert-friendly to novice-friendly.

Compiling, linking

```
   preprocessing   compiling         linking   executing

 header      source      object       library

   a.h
   b.h   ->    b.c    ->   b.o  ---------|
                                         ----->    a.out (b.exe)
   e.h                                   |            |
   f.h   ->    d.c    ->   d.o  ---------|            |runtime
```

```
                                       |             |
  g.h   ->    g.c    ->   g.o          |             |
  h.h   ->    h.c    ->   h.o    ->   h.a  (h.lib)   |
                                       archive       |
  i.h   ->    i.c    ->   i.o                        |
  j.h   ->    j.c    ->   j.o    ->   j.so (j.dll) --|
                                      shared object
```

First C++ program: hello world

```
$ cat hello.cpp
```

```cpp
#include <iostream>

int main()
{
  std::cout << "hello world" << std::endl;
  return 0;
}
```

Compiling, linking, executing

```
# compile + link
$ g++  hello.cpp

# execute
$ ./a.out
hello world

# compile + link + set warnings on
$ g++ -ansi -pedantic -Wall -W hello.cpp

# c++11 mode
$ g++ -std=c++11 -ansi -pedantic -Wall hello.cpp

# set output name to hello.exe
$ g++ -std=c++11 -ansi -pedantic -Wall hello.cpp -o hello.exe

# compile only
```

```
$ g++ -c  hello.cpp
$ ls
hello.o

# will call the linker
$ g++ hello.o
$ ls
a.out

# calls the compiler for all sources then calls the linker
$ g++ a.cpp b.cpp d.o e.a f.so
```

Compiler errors, warnings

If we make a syntax error, the compiler emits error(s):

```
/*
 *  BAD VERSION !!!
 *  Missing semicolon
 */
#include <iostream>

int main()
{
  std::cout << "hello world" << std::endl // missing ;
  return 0;
}
```

```
$ g++ -ansi -pedantic -W -Wall hello.cpp
hello.cpp: In function ‘int main()’:
hello.cpp:10:3: error: expected ‘;’ before ‘return’
   return 0;
   ^
```

If there is a *syntax error*, do compiler do not generate object code. When we have a *warning*, the compiler does generate object output.

```
#include <iostream>

int main()
```

```
{
  int i = 1;
  return 0;
}
```

```
g++ -ansi -pedantic -Wall -W unused.cpp
unused.cpp: In function ‘int main()’:
unused.cpp:9:7: warning: unused variable ‘i’ [-Wunused-variable]
   int i = 1;
       ^
```

Warnings can be serious things in C++, you should treat them as errors unless you are absolute sure in the opposite. Even there, it is a good habit to write warning-free code.