C++: Practical session 1

1 My First C++ Program

Open your favourite text-editor, and type in the following:

```
#include <iostream>

int main(void)
{
  std::cout << "Hello World" << std::endl;
  return 0;
}
```

Save the file as `HelloWorld.C` and then, at the command-line, type the following:

```
g++ HelloWorld.C -o HelloWorld
./HelloWorld
```

You should see `Hello World` displayed on the screen.

2 Compiler and C++ behaviour

The following examples are not meant to put you off using the powerful optimization facilities available in the compiler; they are only intended as a cautionary tale. You do not need to understand the code given at this stage (although you should in a few lectures' time).

2.1 Integer Overflow

Type in the following program into the file `IntOverflow.C`:

```
#include <iostream>

int main(void)
{
  signed int s = 0;
  for(unsigned int i=0 ; i < 10 ; i++)
  {
    for(unsigned int j=0 ; j < 1048576 ; j++)
    {
      s += 1024;
    }
    std::cout << "s = " << s << std::endl;
  }
  return 0;
}
```

Before compiling and running it:

1. What do you think it will do?
2. What *should* it do?

Now compile and run. What happens? Are you surprised?

Now compile using the following command: `g++ IntOverflow.C -o IntOverflow -O2`
This compiles using the second level of optimization.

Now run the code again. What happens? Are you surprised?

Note: This example only produces odd behaviour on `gcc` versions >= 4.8 and at least up to `gcc` 9.1.0
Other compilers may or may not produce unexpected behaviour.

2.2 Integer overflow - clang

If you are using a Mac, or are using `clang++` on Linux, try the following, with either no optimization or `-O2` optimization:

```
#include <stdio.h>

void f1(void) {
  for(int i = 0; i >= 0; i++) {
  }
}

void f2(void) {
  puts(``Formatting /dev/sda1...'');
}

void (*p1)(void) = f1;
void (*p2)(void) = f2;

int main(void) {
  p1();
  return 0;
}
```

(Example taken from `https://twitter.com/m13253/status/1371615680068526081?lang=en-GB` and `godbolt.org/z/1q1bjn`).

This example only produces odd behaviour on `gcc` <= 7.4.0 but also on `clang` at least up to `7.0.1`.

2.3 Float to integer truncation

Type in the following program into the file `FloatIntConversion.C`

```
#include <iostream>

int main(void)
{
  float c = 1e34;
  std::cout << c << std::endl;
  int b = c;
  std::cout << b << std::endl;
  return 0;
}
```

Before compiling and running it:

1. What do you think it will do?
2. What *should* it do?

Now compile and run. What happens? Are you surprised?

Now compile using the following command: `g++ FloatIntConversion.C -o FloatIntConversion -O2`
This compiles using the second level of optimization.

Now run the code again. What happens? Are you surprised?

2.4 Important note

The preceding problems can only arise because they involve *undefined* behaviour within C++. Ordinarily, optimization should not alter the outcome of a program. If it does, then either you are relying on undefined behaviour, or the compiler writers have made a mistake. The former is substantially more likely.

3 Division of negative integers

Find out what you get if you divide -5 by -2 (or similar).

Note that turning optimization on will *not* change the behaviour of the code; the result is implementation defined, not undefined.

Check that `(a/b)*b + a%b == a` as required.