Lecture 3

- Compiling with g++
- Basic C++ building blocks
- Binary arithmetic
- Simple types

How g++ roughly works

- g++ is a C++ compiler
- **g++ -o hello hello.c** checks whether **hello.c** is a valid C++ program,
- if OK, it generates an assembly language representation of **hello.c** in file **hello.s**,
- calls **assembler** (/usr/bin/as) which produces object file **hello.o**, and
- finally calls the **linker** (/usr/bin/ld) which generates executable file **hello**

Assembly Language

- Readable representation of programs the CPU understands (sequence of numbers)
- Issue **g++ -S hello.c** to create **hello.s**
- Use **less hello.s** to see what's in the file
- Assembly language is rarely used anymore
- Only if compiler generates slow/buggy code or the compiler does not make use of the latest CPU instruction set extensions (e.g. SSE)

Basic Program Building Blocks

```
// this is a comment
#include <iostream>   // preprocessor command

int foo(int x)    // function declaration
{                 // block
  return x+1;     // return expression value
}

int main()     // this is where all C programs start
{
  int i = 0;   // variable declaration + initialization
  while (i < 10) {         // loop
    i = i+1;               // expression + assignment
    cout << foo(i) << endl;  // operators +
  }                          // function call
  if (i >= 10) { i = 1; }    // condition
  else         { i = 0; }
  return i;                  // return result, exit
}
```

Identifiers

used for variable, function, member, and label names

- Identifiers are case-sensitive
- Start with _ or letter
- Remaining part all letters, digits, or _s
- Exceptions are C++ keywords such as **`if,then,else, for,while...`**

Valid Identifiers:

sumOfValues x0
FooBar foobar
_x_y_z

Invalid Identifiers:

0x $y .name while
@abc foo#
^_^ ;-)

Comments

- `// this is a single line comment`
- ```
  /* this is a
       multi-
       line
       comment
  */
  ```
- Multi-line comments cannot be nested: **not allowed:** /* /* */ */
- Ample comments in your programs are important – for others and yourself!

Number Systems

- Base is arbitray: unary (1), binary (2), ternary (3), octal (8), decimal (10), hexadecimal (16), ...
- Binary number system
  - Digits 0 and 1 (**bit**)
  - **Byte** = sequence of **8 bits** (contents of one memory cell)
- Integers are represented as sequence of bits
  - One byte can hold 256 different values
  - $00000000_2 = 0_{10}$ $00000001_2 = 1_{10}$
  - $00000010_2 = 2_{10}$ $00000011_2 = 3_{10}$ ...
  - $11111110_2 = 254_{10}$ $11111111_2 = 255_{10}$

Binary Arithmetic

- Instead of digits 0..9, we now only have 0,1
- Arithmetic is done analogously. E.g.
  1+1 = 10, 10+1 = 11, 11+1 = 100, ...
- ```
    1010111     87
  +    1001  +  9
  ----1 1 1 1--  -1--
    1100000     96
  ```
- Weight for each digit is power of 2, rather than power of 10
- For more details follow the link on my webpage (material will be covered in homework 1)

Simple Types

- **bool**: false, true 1 byte (8 bits)
- **char**: ASCII character 1 byte integer
- **short**: -32,768..32,767 2 byte integer (16 bits)
- **int**: -2,147,483,648..2,147,483,647 4 byte integer (32 bits)
- **float**: ~ -10**38..-10**-38,0,+10**-38..+10**38 4 byte fp (7 digits)
- **double**: ~ -10**308..-10**-308,0,+10**-308..+10**308 8 byte fp (15 digits)
- **long double**: ~ -10**4932..-10**-4932,0,+10**-4932..+10**4932 12 byte fp (19 digits)

```
int numOfBeans;
unsigned short sixteenBits;
float PI=3.1415926535;
```


Integer Type Qualifiers `signed,unsigned`

- signed char: **-127..128** 1 byte integer
- unsigned char: **0..255** 1 byte integer
- unsigned short: **0..65535** 2 byte integer
- unsigned int: **0..(2**32)-1** 4 byte integer

```
unsigned char foo = 255;
foo = foo + 1;
// foo is now 0 !
```

Unsigned integers obey laws of arithmetic modulo 232!**

Value Range of Variables

- **IMPORTANT**: make sure variables have the right type to avoid underflows and overflows
- In C++, integer expressions are **not checked** for overflows!

Integer Constants

- An integer constant like **`12345`** is an **`int`**
- Unsigned constants end with **`u`** or **`U`**
- Leading **`0`** (zero) indicates an **octal** (base 8) constant (e.g. **`037`** = 31)
- Leading **`0x`** means **hexadecimal** (base 16, digits 0..9 and a..f are called **nibbles**).
  E.g. **`0x1f`** = 31, **`0x100`** = 256, **`0xa`** = 10

```
int foo = 1234;
unsigned short bar = 60000u;
unsigned char maxChar = 0xff;
```