Tutorial Sheet 3  
ESC101 – Fundamentals of Computing

**Revision (ask for doubts)**

1. Mixed-type expressions (int \* float, int \* long etc)
2. Use of format specifiers in floats %f vs %0.1f and %e vs %0.2e
3. math.h and various functions therein abs, fabs, sqrt, floor, ceil
4. Statements vs expressions

c = a + b is an expression

a + b is also an expression

a is also an expression

**a + is not valid expression**

**c = is not a valid expression**

c = a + b; is a statement

printf(“%d”,a); is a statement

a; is a statement

1. Expressions are evaluated according to the BODMAS rule

Two new rules: unary negation at the top (above brackets) and assignment (below everything else).

1. Mathematical expressions generate numerical values

int a, b = 2, c = 3;

b + 7 will generate value 9

b + c will generate value 5

b will generate value 2

a = 2 \* b will generate value 4

printf(“%d”,a = 2 \* b); will print 4

This is how expressions like p = q = r = 7 are evaluated.

1. if condition with single clause condition if (a < 10) if (b > 20) etc

**Sample Questions to discuss**

**Evaluation order of the expression x = - 5 \* 4 / 2 \* 3 + - 1 \* 2 – 6 \* 3**

1. **Highest**: unary negation

x = **(**- 5**)** \* 4 / 2 \* 3 + **(**- 1**)** \* 2 – 6 \* 3

1. **Next highest**: multiply, division, among them, left associativity

x = **(((**(- 5) \* 4**)** / 2**)** \* 3**)** + **(**(- 1) \* 2**)** – **(**6 \* 3**)**

1. **Next highest**: add, subtraction, among them, left associativity

x = **((**((((- 5) \* 4) / 2) \* 3) + ((- 1) \* 2)**)** – (6 \* 3)**)**

1. **Lowest**: assignment

**(**x = ((((((- 5) \* 4) / 2) \* 3) + ((- 1) \* 2)) – (6 \* 3))**)**

**Compute the factorial of 20 (does not fit inside int but fits inside long).**

#include<stdio.h>

int main(){

long prod = 20\*19\*18\*17\*16\*15\*14\*13\*12\*11\*10\*9\*8\*7\*6\*5\*4\*3\*2\*1;

printf(“%ld”, prod);

return 0;

}

The above program does not work (integer multiplication going on). Need to typecast explicitly. However, doing so in the wrong manner

prod = 20\*19\*18\*17\*16\*15\*14\*13\*12\*11\*10\*9\*8\*7\*6\*5\*4\*3\*2\*(long)1;

also does not work ☹ due to associativity rules. Left to right bracketing and so first 20\*19, 380\*18 etc. calculations are done as integers. By the time the typecast (long)1 is encountered, damage already done.

prod = (long)20\*19\*18\*17\*16\*15\*14\*13\*12\*11\*10\*9\*8\*7\*6\*5\*4\*3\*2\*1;

works since the very first sub-expression to be evaluated is (long)20\*19 which produces a long due to automatic typecasting. Thereafter, Mr C automatically typecasts everything else to long.

**Some Pitfalls and recognizing compiler error messages**

1. Typecasting floats to int will cause loss of digits after decimal

float a = 2.5; int b = (int)a; // b = 2

1. Typecasting int to float or long to float may also cause distortion

long c = 9765432123; a = (float)c; // 9765432320.000000

1. Rounding-errors in integer division can accumulate
2. (a + b)/2 vs a/2 + b/2
3. Only ( ) are valid brackets in math expressions/formulae. [ ] and { } are not valid brackets for math expressions.
4. Not properly bracketing if condition statements may cause unexpected results.