Printing in C

- **Printing value of the built-in data types**
  - **Preliminary:**
    - In **printing**, **C** will view the **content** of a **variable** as a (generic) **sequence of bits**
      - **C** does **not** know (nor ***care***) about the **data type** of the **variable**
    - **You** must tell (instruct) **C** on **how** to ***interpret*** the **bit pattern** !!!
  - The `printf()` function is used to print **values** of ***all* built-in data types** in **C**.
  - **Syntax** of the `printf()` function:

```
printf ( " format string " , value1, value2, ..... );
```

The `"format string"` contains **instructions** on ***how*** to ***interpret*** each of the **values** in the **parameter list**

  - **Format string:**
    - The **format string** in the `printf()` function contains **formatting characters** that instruct the **C compiler** to print a **value** in the given **format**
  - **Commonly used formatting characters:**

Formatting character	Meaning
`%d`	Print the **(next) value** as a ***signed* integer value**
`%u`	Print the **(next) value** as a ***unsigned* integer value**
`%ld`	Print the **(next) value** as a **long *signed* integer value**
`%lu`	Print the **(next) value** as a **long *unsigned* integer value**
`%f`	Print the **(next) value** as a **floating point value**
`%lf`	Print the **(next) value** as a **double precision floating point value**
`%c`	Print the **(next) value** as a **character (i.e., used the ASCII code)**
`%s`	Print the **(next) value** as a **string (will be explained much later)**

  - **Example:**

```
int main( int argc, char* argv[] )
{
   int i = 65, j = 'B';       /* ASCII code for 'B' = 66 */
   float x = 65.0;
   int *p;                    /* Help variable to print x as int */

   printf( "signed integer i: %d\n", i );
   printf( "signed integer j: %d\n", j );
   printf( "signed integer i as character (ASCII code): %c\n", i );
   printf( "signed integer j as character (ASCII code): %c\n", j );

   printf( "\n" );
   printf( "float x: %f\n", x );

   p = (int *)&x;
   printf( "float x as signed integer: %d\n", *p );
}
```

Output:

```
signed integer i: 65
signed integer j: 66
signed integer i as character (ASCII code): A
signed integer j as character (ASCII code): B

float x: 65.000000
float x as signed integer: 1079001088
```

- **Example Program:** (Demo above code) *Example*
  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc printf1.c`
- To run: `./a.out`

- **Warning**
  - **Warning:**
    - The **C compiler** do ***not*** perform any **type checks** in the **printf( )** function call
    - **Therefore**, **you** must **make sure** that the **data type** of the **variables** **correspond** to **formatting character**

- **Example:**

  - **Recall** that **float** uses the **IEEE 754** representation

  - The **representation** for **2.0** is:

```
  2.0(10)  =  10.0(2)  (binary)
           = 10.0(2)   exp 0
           = 1.0(2)    exp 1

  Stored Mantissa = .00000000000000000000000
  Stored Exponent = 01111111 + 1 = 10000000       (excess 127)

The IEEE 754 representation of 2.0 is:

       0 10000000 00000000000000000000000
 Or:
       01000000000000000000000000000000
```

  - The **statement**:

```
  float x = 2.0;

  printf("%d\n", x);
```

  will print ***absurd*** **things**....

- **Example Program:** (Demo above code)

  - Prog file: click here

How to run the program:

- **Right click** on link and **save** in a scratch directory
- To compile: `gcc printf2.c`
- To run: `./a.out`