COL226: Programming Languages

Assignment: The `BigInt` Package

The `factorial` function that we wrote does not allow us to compute any thing more than the factorial of 12 (= 479001600). It gives an overflow exception as soon as the value of factorial crosses the maximum permissible integer(`valOf(Int.maxInt)` which is `1073741823`). It would be nice if one could confidently assert[1] that the factorial of 1000 is

```
402387260077093773543702433923003985719374864210714632543799910429938512
398629020592044208486969404800479988610197196058631666872994808558901323
966994459099742450408707375991882362772718873251977950595099527612087497
462497043601418278094646496291056393887437886487337119181045825783647849
770124766328898359557354325131853239584630755574091142624174743493475534
864657661166779739666882029120737914385371958824980812686783837455973174
136085379534524221586593201928090878297308431392844403281231558611036976
013573042161687476096758713483120254785893207671691324484262361314125087
020800026168315102734182797770478463586817016436502415369139828126481021
092761244896359928705114964975419909342221566832572080821333186116851536
158365469840467089756029009505376164758477284218896796462449451607653534
819890138544248798495995331910172335555660213945039973628075013783761530
127761926849034352625200015888535147331611702103968175921510907788019393
781141945452572238655414610628921879602238389714760885062768629671466746
756291123408243920816015378088989396451826324367161676217916890977991190
754031274622289988005195444414282012187361745992642956581746628302955570
990243241531816172104658320367869061172601587835207515162842255402651704
330422614397428693306169089796848259012545832716822645806652676995865268
272807075781391858178889652208164348344825993266043367660176999612831860
883861502794659551311565520360939881806121385586003014356945272242063446
317974605946825731037900840244324384656572450144028218852524709351906209
023136493273497565513958720559654228749774011413346962715422845862377387
382304838656889764619273838149001407673104466402598994902222217659043399
018860185665264850617997023561938970178600408118897299183110211712298459
641921068884387121855646124960798722908519296819372388642614839657382291
231250241866493531439701374285319266498753372189406942814341185201580141
334482801505139969429015348307764456909907315243327828826986460278986432
139083506217095002597389863554277196742822248757586765752344220207573630
694988250879689281627538488633969099598262809561214509948717012445164612
037902930912088908694202851064018215439945715680594187274899809425474217
582401063677404595741785160829230135358081840096996372524230560855903700
242712434169090041536901059339838357779394109700277534720000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000
```

Hence it is necessary to be able to deal with large integers if the computer has to be a useful tool. The problem of programming arbitrary large and arbitrarily small integers can be split into two sub-problems to be solved as follows.

1. First program arbitrarily large unsigned integers in decimal as a complete package. The signature of this package is as follows:

[1] I am asserting it!

```
signature BIGNAT =
  sig
    type bignat
    exception overflow
    exception underflow
    val zero : bignat
    val normalize : bignat -> bignat
    val fromString : string -> bignat
    val toString : bignat -> string
    val ++ : bignat * bignat -> bignat
    val succ : bignat -> bignat
    val min : bignat * bignat -> bignat
    val max : bignat * bignat -> bignat
    val ** : bignat * bignat -> bignat
    val compare : bignat * bignat -> order
    val << : bignat * bignat -> bool
    val <<= : bignat * bignat -> bool
    val >> : bignat * bignat -> bool
    val >>= : bignat * bignat -> bool
    val == : bignat * bignat -> bool
    val len : bignat -> int
    val lenCompare : bignat * bignat -> order
    val lenLt : bignat * bignat -> bool
    val lenLeq : bignat * bignat -> bool
    val lenGt : bignat * bignat -> bool
    val lenGeq : bignat * bignat -> bool
    val lenEq : bignat * bignat -> bool
    val -- : bignat * bignat -> bignat
    val pred : bignat -> bignat
    exception division_by_zero
    exception emptyList
    val %% : bignat * bignat -> bignat * bignat
    val quo : bignat * bignat -> bignat
    val rem : bignat * bignat -> bignat
  end
```

Most of the functions in this signature are self-explanatory and correspond to the functions given in the `Int` package of SML. The only addition are all those functions relating to the "length" of an arbitrarily large natural number, since it can sometimes enable easy comparisons between unsigned numbers in normal form (i.e. numbers which have no leading zeroes). **Implement the structure**

```
structure Bignat:  BIGNAT
```

2. Then use the functor feature of SML to program large integers using the package of large unsigned numbers to implement the following functor

```
functor BigInt (Bn:BIGNAT):
  sig
    type bigint
    val bigzero: bigint
    val normalize : bigint -> bigint
    val bigint: int -> bigint
    val fromString : string -> bigint option
    val toString : bigint -> string
    val bigint : int -> bigint
    val int : bigint -> int option
    val ~~ : bigint -> bigint
```

```
  val abs : bigint -> bigint
  val ++ : bigint * bigint -> bigint
  val succ : bigint -> bigint
  val min : bigint * bigint -> bigint
  val max : bigint * bigint -> bigint
  val sign : bigint -> int
  val sameSign : bigint * bigint -> bool
  val ** : bigint * bigint -> bigint
  val compare : bigint * bigint -> order
  val << : bigint * bigint -> bool
  val <<= : bigint * bigint -> bool
  val >> : bigint * bigint -> bool
  val >>= : bigint * bigint -> bool
  val == : bigint * bigint -> bool
  val len : bigint -> int
  val lenCompare : bigint * bigint -> order
  val lenLt : bigint * bigint -> bool
  val lenLeq : bigint * bigint -> bool
  val lenGt : bigint * bigint -> bool
  val lenGeq : bigint * bigint -> bool
  val lenEq : bigint * bigint -> bool
  val -- : bigint * bigint -> bigint
  val pred : bigint -> bigint
  exception division_by_zero
  val %% : bigint * bigint -> bigint * bigint
  val div : bigint * bigint -> bigint
  val mod : bigint * bigint -> bigint
  val quo : bigint * bigint -> bigint
  val rem : bigint * bigint -> bigint
end (* sig *)
```