COMP2012H Assignment 2 Report

Author: WONG Yuk Chun (20419764)

Task 1

the link list is from most significant digit to least significant digit with sign as head.

iostream operators

a. `<;<;` **:**

it is just similar to the `to_string` function, loop through the link list and print, then return ostream object passed.

b. `>;>;` **:**

it is just calling the `from_string` function using the input string as parameter, then return istream object passed.

Comparison operators

a. `a == b` **:**

it is just looping through the two link list on the same time, once there is a different character at the same corresponding position of the two lists, return false. If the list have different length, return false. else it is true.

b. `a != b` **:**

it is inversing the result of `a==b`

c. `a >; b` **:**

if a is positive and b is negative, a of course bigger than b, true. if a is negative and b is positive, a of course smaller than b, false. if both are positive, if index of dot in a (from LHS) > that of b, a of course bigger than b, true. if index of dot in a (from LHS) < that of b, a of course smaller than b, false. if length of a > that of b, a of course bigger than b, true. if length of a < that of b, a of course smaller than b, false. the exception case is same length and same digit places are present in a and b check from most significant digit to least digit by digit, once $a_i > b_i$ =>true $a_i < b_i$ =>false till the end no result=>they are the same=>false if both are negative, the return result is the inverse of result of both are positive.

d. `a <; b` **:**

equivalent with `b >; a`

e. `a >;= b` **:**

equivalent with inverse of `b >; a`

f. `a <;= b` **:**

equivalent with inverse of `a >; b`

Assignment operator

a. `=`**:**

remove all nodes in the link list, construct new link list by copying the value of the source link list.

Arithmetic operators

a. `a + b` **:**

1. compare the absolute value of a and b, let a be the bigger and b be the smaller.
2. align the dot position of a and b by adding zero at front and at the back, like
   a = XXXXXX.XX
   b = X.XXXXX
   will become
   a = XXXXXX.XX000
   b = 00000X.XXXXX
3. loop from least significant figure to the most, if same sign, preform addition, else preform subtraction, increment or decrement LHS digit correspondingly when necessary.
4. finally return the value with sign of a

b. `a - b`**:**

1. flip the sign of copy of b, b', '+' to '-' or '-' to '+'
2. return value of a+b'.

c. `a * b`**:**

1. remove the dot of absolute value of a and b, namely `a';` and `b';`.
2. if b' equals to 10, semi-product is appending `0` to `a';`, a base case for recurrence relationship of `*`
   if a' equals to 10, semi-product is appending `0` to `b';`, a base case for recurrence relationship of `*`
   if b' is one digit, semi-product (initially = 0) is the loop from first digit of a', each time multiply by 10 and add the multiple of the digit and b.
   else b' is more than one digit, semi-product is a' * last digit of b' + 10* a*remaining digits of b'
3. add dot to the semi-product, location from right is the sum of number of decimal digits of a and b.
4. set sign of the semi-product, if sign of a and b are the same, sign is +, if not sign is -, and now it is the final product.

d. `a / b`**:**

1. preprocess:
   1. if b equals to 0, throw division by zero error
   2. let absolute of `a` and `b` be `a';` and `b';` respectively.
   3. calculate the resultant precision as 1+ max of a's precision and b's precision.
   4. make a' to be $a' * 10^{precision}$ so that the quotient will have extra digits (calculating the decimals).
2. recursion:
   1. if a<b terminate 2. with 0
   2. else

1. calculate the more significant quotient, p, by calling a/(b*10)
2. calculate the remainder by the more significant quotient, $r = a' - b * p * 10$
3. let the quotient generated by this recursion level is q, while $r >= b'$, subtract b' from r and q increment by 1.
4. finally leave 2. with the combined quotient of this recursion level and previous levels, $q + p * 10$

3. add back the sign and decimal point to the rounded result and return.

e. `a ^ b`**:**

1. if b<0, it is `a/abs(b)` with precision of max of that of a and b, and round off accordingly.
2. if b equals 0, result is 1
3. if b equals 1, result is a
4. if b equals 2, result is a*a
5. else
    1. turn b to a bit-string, set `result` as 1.
    2. for each digit b_i in b, if b_i is 1, `result` multiply by a^{2^i}
    3. return result

f. increment and decrement

`++a` : add 1 to this and return this

`a++` : make a copy of a, add 1 to this, return the copy

`--a` : minus 1 from this and return this

`a--` : make a copy of a, minus 1 from this, return the copy

Task 2

The link lists will have dummy heads.

1

3 global node pointers in total:

a. temp: main accessing pointer

b. p: temp's parent node

c. M: pointing to the $m - 1^{th}$ node

temp traverse the link list from 1^{st} data node to last data node, p start from the dummy head.

when temp reach m^{th} node, record M as p

before temp reach n^{th} node, remove the node which temp is pointing and insert it to where M is pointing, and continue with next node. After that, make temp pointing to pointee of p's next and continue.

when temp reach a NULL, print out the resultant link list

2

3 global node pointers in total:

a. m: main accessing pointer

b. h: temp's parent node

c. n: m's pointee's next node

temp traverse the link list from 1^{st} data node to last data node, h start from the dummy head, n start from 2^{nd} data node.

if it sees m data equals to n data, delete n and make the next node as n, repeat this until m not equal to n and remove m. use p's next as m and p's next's next as n and continue.

when m or n reach NULL, print out the resultant link list.

3

2 global node pointers in total:

a. m: main accessing pointer

b. h: temp's parent node

temp traverse the link list from 1^{st} data node to last data node, h start from the dummy head

record current m's next as n, and n's next is k.

reverse m and n by making h's next as n, n's next as m, m's next as k.

now it change from h->m->n->k to h->n->m->k

if k is actually null, terminate and output the link list.

take h as m and m as m's next and continue.

if m and m's next reach NULL, print out the resultant link list.