```
/*-----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
file: ../str/str.cpp ulongnum.cpp ulongnumtest.cpp

On linux:
g++  ../str/str.cpp  ulongnum.cpp ulongnumtest.cpp
valgrind a.out
-- All heap blocks were freed -- no leaks are possible

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test ulongnum object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "ulongnum.h"

/*--------------------------------------------------------
static definition - only once at the start
Change to false, if you don't need verbose
--------------------------------------------------------*/
template <typename T>
bool darray<T>::_display = false;

bool str::_display = false;
bool ulongnum::_display = false;

/*--------------------------------------------------------
local to this file. Change verbose = true for debugging
--------------------------------------------------------*/
static bool verbose = false;

/*-----------------------------------------------------------------
test multiplication
-----------------------------------------------------------------*/
static void test_multiplication() {
  ulongnum a(789);
  cout << "a = " << a << endl;
  ulongnum b("56");
  cout << "b = " << b << endl;
  ulongnum ans = a * b;
  cout << "ans = " << ans << endl;
  assert(ans == 44184);

  ulongnum rsa129(                                                          ↙
    "1143816257578888676692357799761466120102182967212423625625618429357069352457338978305971235639587050589890751475992900268795435 41");
```

```
  ulongnum rsa129(
    "114381625757888867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541");
  ulongnum p1("3490529510847650949147849619903898133417764638493387843990820577");
  ulongnum p2("32769132993266709549961988190834461413177642967992942539798288533");
  ulongnum p1p2 = p1 * p2;
  cout << "p1 = " << p1 << endl;
  cout << "p2 = " << p2 << endl;
  cout << "p1p2 = " << p1p2 << endl;
  assert(p1p2 == rsa129);
}

/*-----------------------------------------------------------------
test addition
-----------------------------------------------------------------*/
static void test_addition() {
  ulongnum a(9789);
  ulongnum b("100000");
  ulongnum c('7');
  cout << "a = " << a << endl;
  cout << "b = " << b << endl;
  cout << "c = " << c << endl;
  ulongnum sum = a + 78 + b + c;
  cout << "sum = a + 78 + b + c = " << sum << endl;
  assert(sum == 109874);
}

/*-----------------------------------------------------------------
test basic
-----------------------------------------------------------------*/
static void test_basic() {
  ulongnum a(789);
  cout << "a = " << a << endl;
  ulongnum b("56789");
  cout << "b = " << b << endl;
  ulongnum c("1234567890123456789012345678901234567890123456789012345678901234567890");
  cout << "c = " << c << endl;
  ulongnum ta(a);
  cout << "ta = " << ta << endl;
  ta = b;
  cout << "ta = " << ta << endl;
}

/*-----------------------------------------------------------------
test factorial
-----------------------------------------------------------------*/
static void test_factorial() {
  // http://puzzles.nigelcoldwell.co.uk/nineteen.htm
  {
    const char* fact100 =                                                                      ↙
    "93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518 ↙
    286253697920827223758251185210916864000000000000000000000000";
    clock_t start = clock();
    ulongnum c100;
    c100.factorial(100);
    cout << "Factorial of 100 = " << endl;
    cout << c100 << endl;
    assert(c100 == fact100);
    clock_t end = clock();
    double d = double(end - start) / CLOCKS_PER_SEC;
    cout << "Run time for !100 = " << " is " << d << " secs" << endl;
  }

  //http://justinwhite.com/big-calc/1000.html
  {
  const char* fact1000 =                                                                       ↙
    "402387260077093773543702433923003985719374864210714632543799910429938512398629020592044208486969404 ↙
    4800479988610197196058631666872994808558901323829669944590997424504087073759918823627727188732519779 ↙
    9505950995276120874975462497043601418278094646496291056393887437886487337119181045825783647849977012 ↙
    4766328898359557354325131853239584630755574091142624174743493475534286465766116677973966688202912073 ↙
    7914385371958824980812686783837455973174613608537953452422158659320192809087829730843139284440328 ↙
    1231558611036976801357304216168747609675871348312025478589320767169132448426236131412508780208000026 ↙
    1683151027341827977704784635868170164365024153691398281264810213092761244896359928705114964975419909 ↙
    3422215668325720808213331861168115536158365469840467089756029009505376164758477284218896796462449451 ↙
    6076535340819890138544248798495995331910172335555660213945039973628075013783761530712776192684903 ↙
    4352625200015888535147331611702103968175921510907788019393178114194545257223865541461062892187960223 ↙
    8389714760885062768629671466746975629112340824392081601537808898939645182632436716167621791689097799 ↙
    1190375403127462228998800519544441428201218736174599264295658174662830295557029902432415318161721046 ↙
    5832036786906117260158783520751516284225540265170483304226143974286933061690897968482590125458327168 ↙
    2264580665267699586526822728070757813918581788896522081643483448259932660433676601769996128318607883 ↙
    8615027946595513115655203609398818061213855860030143569452722420634463179746059468257310379008402443 ↙
    2438465657245014402821885252470935190620929023136493273497565513958720559654228749774011413346962715 ↙
    4228458623773875382304838656889764619273838149001407673104466402598994902222217659043399018860185665 ↙
    2648506179970235619389701786004081188972991831102117122984590164192106888438712185564612496079872290 ↙
    8519296819372388642614839657382291123125024186649353143970137428531926649875337218940694281 ↙
    1434118520158014123344828015051399694290153483077644569099073152433278288269864602789864321139083500 ↙
    6217095002597389863554277196742822248757586765752344220207573630569498825087968928162753848863396900 ↙
    9959826280956121450994871701244516461260379029309120889086942028510640182154399457156805941872748990 ↙
    8094254742173582401063677404595741785160829230135358081840096996372524230560855903700624271243416900 ↙
    9004153690105933983835777939410970027753472000000000000000000000000000000000000000000000000000000000 ↙
    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 ↙
    00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000";
  clock_t start = clock();
  ulongnum c1000;
  c1000.factorial(1000);
  cout << "Factorial of 1000 = " << endl;
  cout << c1000 << endl;
  assert(c1000 == fact1000);
  clock_t end = clock();
  double d = double(end - start) / CLOCKS_PER_SEC;
  cout << "Run time for !1000 = " << " is " << d << " secs" << endl;
}
}

/*------------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  ulongnum::set_display(verbose);
  test_basic();
  test_addition();
  test_multiplication();
  test_factorial();
  return 0;
}

//EOF


```