**.H**

#ifndef DBIGUINT\_H

#define DBIGUINT\_H

#include <cstdlib>

#include <string>

class dbiguint

{

public:

// pre: none

// post: creates a dynamic bigint value 0

dbiguint();

// pre: s[0], ..., s[s.size()-1] are digits

// post: creates a dbiguint whose digits are given in s

dbiguint(const std::string & s);

/\*We aren't implementing this constructor yet

// pre: none //NEW

// post: copy constructor: creates a new dynamic bigint which is

// a copy of given dynamic bigint

dbiguint(const dbiguint &);

\*/

// pre: none //NEW

// post: returns dynamically allocated memory to heap

~dbiguint();

// pre: none //NEW

// post: makes this dynamic bigint a copy of given dynamic bigint

// void operator =(const dbiguint &);

// pre: none //NEW

// post: returns the size of the memory block of this dbiguint

std::size\_t size() const;

// pre: none

// post: returns the digit at given pos (0 if does not exist)

// pos 0 is the least significant (units) digit

unsigned short operator [](std::size\_t pos) const;

// pre: none

// post: returns 0 if this dbiguint equals given dbiguint

// 1 if this dbiguint > given dbiguint

// -1 otherwise

// int compare(const dbiguint &) const;

// pre: none

// post: returns a string containing the digits and sign of this dbiguint

// std::string toStdString() const;

// pre: none

// post: adds/subtracts given dbiguint to this dbiguint

void operator +=(const dbiguint &);

// void operator -=(const dbiguint &);

// void operator \*=(const dbiguint &);

// pre: none

// post: if newcapacity\_ <= capacity\_ then do nothing (cannot shrink)

// else allocate a new block with size newcapacity\_

// copy existing digits and fill the rest with 0

void reserve(std::size\_t newcapacity\_);

private:

unsigned short \*data\_;

std::size\_t capacity\_;

// INVARIANTS:

// data\_ points to (has the address of) a dynamic array

// of capacity\_ digits

// data\_[0] = least significant (units) digits

// data\_[k] = digit at position k (or 0 if not used)

};

// nonmember functions

/\*

dbiguint operator +(const dbiguint &, const dbiguint &);

dbiguint operator -(const dbiguint &, const dbiguint &);

dbiguint operator \*(const dbiguint &, const dbiguint &);

bool operator < (const dbiguint &, const dbiguint &);

bool operator <= (const dbiguint &, const dbiguint &);

bool operator == (const dbiguint &, const dbiguint &);

bool operator != (const dbiguint &, const dbiguint &);

bool operator >= (const dbiguint &, const dbiguint &);

bool operator > (const dbiguint &, const dbiguint &);

\*/

std::ostream & operator << (std::ostream &out, const dbiguint &b);

//std::istream & operator >> (std::istream &, dbiguint &);

#endif // DBIGUINT\_H

**.CPP**

#include "dbiguint.h"

#include <iostream>

dbiguint::dbiguint(){

data\_ = new unsigned short[1];

data\_[0] = 0;

capacity\_ = 1;

}

dbiguint::dbiguint(const std::string &a){

data\_ = new unsigned short[a.size()];

capacity\_ = a.size();

int j = 0;

char zero = '0';

int codeZero = zero;

for(std::size\_t i = 1; i < a.size()+1; i++){

int tempNum = a[a.length() - i];

int numCode = tempNum - codeZero;

data\_[j]=numCode;

j++;

}

}

std::size\_t dbiguint::size() const{

return capacity\_;

}

unsigned short dbiguint::operator [](std::size\_t pos) const{

assert(pos < capacity\_);

return dbiguint::data\_[pos];

}

std::ostream& operator <<(std::ostream &out, const dbiguint &b){

for(size\_t i = 0; i < b.size(); i++){

out<<b[i]<<" ";

}

out << std::endl;

return out;

}

void dbiguint::reserve(std::size\_t newcapacity\_){

if(newcapacity\_ > capacity\_){

unsigned short \* tempData\_ = new unsigned short[newcapacity\_];

for (size\_t i = 0; i < capacity\_; i++){

tempData\_[i] = data\_[i];

}

for(size\_t j = capacity\_; j < newcapacity\_; j++){

tempData\_[j] = 0;

}

capacity\_ = newcapacity\_;

delete [] data\_;

data\_ = tempData\_;

tempData\_ = nullptr;

}

}

void dbiguint::operator +=(const dbiguint & b){

if(b.size() > capacity\_){

reserve(b.size());

}

else{

reserve(capacity\_);

}

for(size\_t i = 0; i < capacity\_; i++){

if (data\_[i] + b.data\_[i] > 9){

data\_[i+1] = data\_[i+1] + 1;

int tempHolder = data\_[i] + b.data\_[i];

tempHolder = tempHolder%10;

data\_[i] = tempHolder;

}

else{

data\_[i] = (data\_[i] + b.data\_[i]);

}

}

}

dbiguint:: ~dbiguint(){

delete [] data\_;//This is the only piece we NEED

data\_ = nullptr;

capacity\_ = 0;

}

**MAIN.CPP**

#include "dbiguint.h"

#include <iostream>

using namespace std;

int main(){

string str = "88";

string str2 = "2";

dbiguint defaultConst = dbiguint();

dbiguint usefulConst = dbiguint(str);

dbiguint coolBeans = dbiguint(str2);

// cout << usefulConst[2]<<endl;

// cout << usefulConst<<endl;

cout<< "After" << endl;

usefulConst+=(coolBeans);

//usefulConst.reserve(20);

cout << usefulConst<<endl;

return 0;

}