OOP Assignment

(inf\_int Class)

Team #8

유용민

설지환

안지완

이의제

이주형

채승운

최동욱

Index.

* Introduction
* Concept of the project
* Required Components
* Specification of the functions
* Implementation
* Execution Results
* Conclusion

List of team members

|  |  |  |
| --- | --- | --- |
| Name | ID | Role |
| 유용민 |  |  |
| 설지환 |  |  |
| 안지완 |  |  |
| 이의제 |  |  |
| 이주형 |  |  |
| 채승운 |  |  |
| 최동욱 |  |  |

Introduction

Our team’s project is mainly about “making a class that can maintain an infinite length of number, having a basic operator of addition, subtraction, and multiplication.” As a basic ‘Integer class’ given by C++ can only maintain at most approximately 2 billion, we should create a customized integer class, named as ‘inf\_int.’ In this report, we will explain how we made a brief concept, specification, and implementation of this customized ‘inf\_int’ class.

Concept of the ‘inf\_int’

As we explained above, a basic integer class (a.k.a ‘int’) can only maintain at most 2 billion, which means that assigning above 2 billion to the ‘int’ will make an overflow of the program, leading to the fatal error to the program. To prevent this, we decided to use ‘string’ instead. A string type can maintain an infinite length of the string (actual length is 4.2billion, which is still close to infinite, 2^(4.2billion)). A brief abstraction of the class ‘inf\_int’ looks like this:

class inf\_int

{

private :

char\* digits;

unsigned int length;

bool thesign;

public :

inf\_int();

inf\_int(int);

inf\_int(const char\* );

inf\_int(const inf\_int&);

~inf\_int();

inf\_int& operator=(const inf\_int&);

friend bool operator==(const inf\_int& , const inf\_int&);

friend bool operator!=(const inf\_int& , const inf\_int&);

friend bool operator>(const inf\_int& , const inf\_int&);

friend bool operator<(const inf\_int& , const inf\_int&);

friend inf\_int operator+(const inf\_int& , const inf\_int&);

friend inf\_int operator-(const inf\_int& , const inf\_int&);

friend inf\_int operator\*(const inf\_int& , const inf\_int&);

friend ostream& operator<<(ostream& , const inf\_int&);

};

A customized class ‘inf\_int’ has a responsibility to not only operate basic arithmetic functions properly, but also perform comparison operator and I/O functions. Thus, we overrode the basic operators to get our ‘inf\_int” class as the parameters, declaring them as ‘friend’ functions to give them permission to access the internal data (digits, length, thesign). By doing this, operators will access the char-type digits and perform the functions by reading each digit. Now we need to specify the implementations.

Specifying each implementations

Constructors

Before explaining the implementation of the function, we first need to specify the structure of the internal data (digits, length, thesign). As explained above, we should store digits in char type. However, since we cannot figure out the exact length of the number before declaring it, we should not use the static array. Instead, we should use ‘dynamic allocation.’ This concept will be used in the constructor of the ‘inf\_int’ class, as the internal members are initialized when the constructor is called.

Four types of constructors are declared in the ‘inf\_int’ class : One that gets an integer as a parameter, one that gets a string, another as a copy constructor, and the other with none parameter.

Starts with the first (integer), we should beware that the structure of a string type, and the integer type is different. As we assumed that the length of the digits is infinite, we cannot simply cast the integer to the string type. If doing so, it won’t properly perform the arithmetic functions with another ‘inf\_int’ object having infinite length of digits. Considering this case, we should divide the integer with 10 to get each digit, which will be saved in reversed order for the reason that is explained by the diagram below.

15491

-> 1549 … 1 ( Quotient | Remainder)

-> 154 … 9 -> count the iteration in order to set the

-> 15 … 4 memory size of the digit

-> 1 … 5

-> 0 … 1

Char buf[100] reversed order (19451) -> temporarily stores each digit

-> dynamically allocated memory

char\* digit

Dividing 10 repeatedly and extracting each remainder will produce a single digit of the target integer, which will be stored in the temporary char array (buf) but in reversed order. After that, the data of the temporary char array will be moved to the digit, which is initialized with dynamic memory allocation method. What makes such process possible is the loop of the division. Counting the loop iteration that performs division until the quotient becomes zero, will allow the program to get the length of the target integer, and by using the length, we can dynamically allocate the memory with such size and initialize the internal member, “length.” The sign will be equal to the target integer.

The code will be like this:

inf\_int::inf\_int(int n)

{

char buf[100];

if (n < 0)

{

this->thesign = false;

n = -n;

}

else

{

this->thesign = true;

}

int i = 0;

while (n > 0)

{

buf[i] = n % 10 + '0';

n /= 10;

i++;

}

if (i == 0)

{

new (this) inf\_int();

}

else

{

buf[i] = '\0';

this->digits = new char[i + 1];

this->length = i;

strcpy(this->digits, buf);

}

}

Once the constructor with integer parameter is created, the other types of constructor is not that complicated, as the data type is same as the internal member, ‘digit.’ The one with string parameter just needs some process to read each character and copy into the ‘char\* digit,’ getting the length of the target string which will be same as the value of the member ‘unsigned int length,’ and deciding the sign by reading the first character.

Copy constructor only needs a process to read the target ‘inf\_int’ object’s internal member and literally copy each data into the internal members.

A default constructor (that doesn’t need any parameter) will have a simple process to initialize the internal members. In this project, we will initialize the digit as zero, additionally putting the null at the end of the digit array to indicate the end of the string, and set the length as 1 and bool sign as true.

Code will look like this :

inf\_int::inf\_int()

{

this->digits = new char[2];

this->digits[0] = '0';

this->digits[1] = '\0';

this->length = 1;

this->thesign = true;

}

inf\_int::inf\_int(const char \*str)

{

unsigned int i;

if (str[0] == '-')

{

this->thesign = false;

this->length = strlen(str) - 1;

this->digits = new char[length + 1];

for (i = length - 1; i >= 0; i--)

{

digits[i] = str[length - i];

}

}

else

{

this->thesign = true;

this->length = strlen(str);

this->digits = new char[length + 1];

for (i = length - 1; i >= 0; i--)

{

digits[i] = str[length - i - 1];

}

}

}

inf\_int::inf\_int(const inf\_int &a)

{

this->digits = new char[a.length + 1]

strcpy(this->digits, a.digits);

this->length = a.length;

this->thesign = a.thesign;

}

inf\_int::~inf\_int()

{

delete digits;

}