**2023-2 OOP Project #2 Prob01**

폰트, 시계, 그래픽, 로고이(가) 표시된 사진

자동 생성된 설명

|  |  |
| --- | --- |
| **Team 4** | |
| 20194198 | Min-sik Kim |
| 20216793 | Jun-seob Kim |
| 20222663 | Ki-yeong Kweon |
| 50231640 | Illan-Emmanuel COCO-GUIGNARD |
| 50231643 | Marie Brouard |
| 50231647 | Thomas CHEVALME |
| **Speaker** | Illan-Emmanuel COCO-GUIGNARD |

1. **Summary**

This project is designed to handle arithmetic operations for integers of unlimited size, overcoming the typical limitations of primitive data types in programming languages like C++. The project consists of four files:

**inf\_int.h**: This file declares the inf\_int class and its member functions, which encapsulate the functionality for infinite integer operations, ensuring that any integer value, regardless of its length, can be stored and manipulated.

**inf\_int.cpp**: This file contains the implementation of inf\_int class's member functions.

**main.cpp**: This is the driver program that includes the main function.

**Makefile**: This enables compilation through the 'make' command.

1. **Compilation & Execution Method**

**System Requirements**

* OS capable of running C++ compilers (Windows, macOS, Linux).
* C++ compiler supporting the C++11 standard or later.
* Make utility for processing the makefile

**Compilation Instructions:**

To compile the inf\_int project, navigate to the directory containing the source files and run the following command in terminal:

**$ make**

This will execute the instructions defined in the makefile to compile the source files and link the generated object files into an executable.

**Execution Instructions**:

After compilation, you can run the program by run following command in terminal.

**$ ./a.out**

1. **Description on functionality**

This program goes beyond the limits of numerical variable types in a C++ environment and can perform operations if there is enough memory, regardless of the number of digits.

You can perform addition, subtraction, multiplication, and comparison operations, excluding division, and see the results by entering the values and expressions to test in the Main function.

1. **Implementation Details**

**Dynamic Array Storage**

Each inf\_int object contains a dynamically allocated array to store digits in a reversed order, enabling easy carry management.

**Carry-Over Handling**

During addition and subtraction, carry-over between digit positions is managed by loops that process each digit in sequence. If the sum of two digits exceeds 9 during addition, the algorithm carries over the excess to the next higher digit. During subtraction, if one digit is less than another, it borrows from the next higher digit.

**Multiplication using FFT**

To multiply big numbers in efficient way, we use Fast Fourier Transform. FFT splits up the numbers into smaller parts, making the process of multiplying them less complicated and quicker than the approach of traditional multiplication**.**

**Optimizations**

To enhance efficiency, Program check some special cases before operations. For example, In the case of multiplication, the case where one term is one of 0, -1, and 1 is processed by simple way first and the operation is not further performed.

This detailed implementation enables the inf\_int system to perform high-precision arithmetic reliably,

1. **Result of SW system design**

|  |
| --- |
| **Inf\_int** |
| - digits: char\*  - length: unsigned int  - thesign: bool |
| + Add(char, unsigned int): void  + Sub(char \*, char \*): void  + inf\_int()  + inf\_int(int)  + inf\_int(const char\*)  + inf\_int(const inf\_int&)  + ~inf\_int()  + operator=(const inf\_int&): inf\_int&  + <<friend>> operator==(const inf\_int&, const inf\_int&): bool  + <<friend>> operator!=(const inf\_int&, const inf\_int&): bool  + <<friend>> operator>(const inf\_int&, const inf\_int&): bool  + <<friend>> operator<(const inf\_int&, const inf\_int&): bool  + <<friend>> operator+(const inf\_int&): inf\_int  + <<friend>> operator-(const inf\_int&): inf\_int  + <<friend>> operator+(const inf\_int&, const inf\_int&): inf\_int  + <<friend>> operator-(const inf\_int&, const inf\_int&): inf\_int  + <<friend>> operator\*(const inf\_int&, const inf\_int&): inf\_int  + <<friend>> operator<<(ostream&, const inf\_int&): ostream& |

The diagram above represents the access specifiers, argument information, and return types for the fields and methods of the inf\_int class.

**Private Members:**

**digits:** A pointer to a character array where the individual digits of the large integer are stored. Each character represents a single digit.

**length:** An unsigned integer that holds the total number of digits in the number.

**thesign:** A boolean value that holds the sign of the number, where true represents a positive and false is negative.

**Private Methods:**

**Add(char, unsigned int):** A private method used internally to add digits at a specific position in the number.

**Sub(char \*, char \*):** A private method used internally to subtract digits from another.

**Constructors and Methods:**

**inf\_int():** A constructor that initializes an inf\_int object, default value is zero.

**inf\_int(int):** A constructor that converts an int into an inf\_int object.

**inf\_int(const char\*):** A constructor that creates an inf\_int object from a string.

**inf\_int(const inf\_int&):** A copy constructor for creating a new inf\_int object as a copy of an existing one.

**~inf\_int():** A destructor for the dynamic memory allocated for the object.

**operator=(const inf\_int&):** An assignment operator.

**operator==(const inf\_int&, const inf\_int&):** An equality comparison operator.

**operator!=(const inf\_int&, const inf\_int&):** A not-equal comparison operator.

**operator>(const inf\_int&, const inf\_int&):** A greater-than comparison operator.

**operator<(const inf\_int&, const inf\_int&):** A less-than comparison operator.

**operator+(const inf\_int&):** A unary plus operator for creating a positive copy of an inf\_int object.

**operator-(const inf\_int&):** A unary minus operator for creating a negated copy of an inf\_int object.

**operator+(const inf\_int&, const inf\_int&):** An addition operator for adding two inf\_int objects.

**operator-(const inf\_int&, const inf\_int&):** A subtraction operator for subtracting one inf\_int object from another.

**operator\*(const inf\_int&, const inf\_int&):** A multiplication operator for multiplying two inf\_int objects.

**operator<<(ostream&, const inf\_int&):** An insertion operator for printing inf\_int objects value.

1. **Execution results**

**Test cases**

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Test cases use integrator-based, string-based inf\_int objects and objects generated by copy constructor. Inputs within the range of the numerical variable type and inputs beyond the range are mixed and calculated, and the result value is output to the terminal window.

**Result of execution**

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

The picture above shows the terminal window after program is executed through the ./a.out command. It can be confirmed that the results of all inputs and operations are accurately output.

1. **Application of object oriented concepts**

Object-oriented programming (OOP) principles such as encapsulation, abstraction, and operator overloading were employed. From this project, the importance of clean design and the power of OOP in handling complex systems were learned.

1. **Conclusion**

Through this project, we learned how to overcome the limitations of numerical data types and compute numbers with an infinite number of digits. Implementing the program ourselves allowed us to experience many trials and errors, especially in handling strings in C++. As a result, we were able to develop a program capable of calculating very large numbers, as far as the hardware system allows.