



## BIG INT in C

August 15, 2023

Aayush Singh (2020CHB1036) ,  
Neeraj Saini (2020CHB1047) ,  
Rohan Keshari (2020CHB1052)

---

**Instructor:**

Dr. Anil Shukla

**Teaching Assistant:**

Avadhesh Gaur

**Summary:** The primary objective of our project is to find the exact value of  $\pi$  up to 10000 decimal places. We are computing this by using C language. But the range of integers in C language is bounded, So we need to create a data structure which can hold huge integers; we call it - BIG INT. We are implementing all major arithmetic operations in BIG INT to achieve our final objective successfully. Also, We are implementing complex numbers in C and making functions that can deal with substantial fractions. But finding  $\pi$  is itself a mathematical complexity. We are first using Newton-Raphson Method for root convergence and then using Chudnovsky Algorithm to find  $\pi$ .

---

## 1. Introduction

This project is divided into three parts.

- The implementation of our self-designed data structure - BIGINT.
- Making of utility functions & implementing complex numbers and fractions.
- Use of BIGINT to deduce the value of  $\pi$  up to 10000 decimal places.

### I) IMPLEMENTATION OF BIGINT

To implement BIGINT, we are making functions such as BIGINT Addition, BIGINT Subtraction, BIGINT Multiplication, BIGINT Division, BIGINT Decimal Division, BIGINT Remainder (Modulo), BIGINT GCD, BIGINT Power, BIGINT Factorial, BIGINT Square Root. These functions can be called and accessed using the switch case command. These functions will be further internally used in the process of calculation of  $\pi$ .

### II) MAKING OF UTILITY FUNCTION (COMPLEX NUMBERS & FRACTIONS, ETC)

To add more worth to our program. We are adding some utility operations and implementing Complex Numbers & Fractions. To implement complex numbers, we are making functions such as Complex Addition, Complex Subtraction, Complex Multiplication, Complex Division and Complex Conjugate. Furthermore, to implement Fractions, we are making functions such as Fraction Addition, Fraction Subtraction, Fraction Multiplication, Fraction Division, and Fraction- Reduced to Simplest Form.

### III) USING BIGINT TO FIND $\pi$

To find the value of  $\pi$ , we are using Chudnovsky Algorithm. The Chudnovsky algorithm is a fast method for calculating the digits of  $\pi$ , based on Ramanujan's  $\pi$  formulae. To use Chudnovsky Algorithm, we need to use the Newton-Raphson Method of convergence to find the exact value of roots coming in Chudnovsky's formula. We are using all the functions we have made in implementing BIGINT to compute Chudnovsky's Formula & Hence finding the value of  $\pi$  exactly up to 10000 decimal places.

## 2. Equations

We have used two major equations-

- i) Newton-Raphson Method
- ii) Chudnovsky Algorithm

### I) NEWTON-RAPHSON METHOD

The Newton–Raphson method, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a real-valued function. The most basic version starts with a single-variable function  $f$  defined for a real variable  $x$ , the function's derivative  $f'$ , and an initial guess  $x_0$  for a root of  $f$ . If the function satisfies sufficient assumptions and the initial guess is close, then

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

is a better approximation of the root than  $x_0$ . Geometrically,  $(x_1, 0)$  is the intersection of the  $x$ -axis and the tangent of the graph of  $f$  at  $(x_0, f(x_0))$ : that is, the improved guess is the unique root of the linear approximation at the initial point. The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently precise value is reached.

### II) CHUDNOVSKY'S ALGORITHM

The Chudnovsky algorithm is a fast method for calculating the digits of  $\pi$ , based on Ramanujan's  $\pi$  formulae. The algorithm is based on the negated Heegner number  $d = -163$ , the  $j$ -function  $j\left(\frac{1+i\sqrt{163}}{2}\right) = -640320^3$ , and on the following rapidly convergent generalized hypergeometric series.

$$\frac{1}{\pi} = 12 \sum_{q=0}^{\infty} \frac{(-1)^q (6q)! (545140134q + 13591409)}{(3q)! (q!)^3 (640320)^{3q + \frac{3}{2}}}$$

For a high-performance iterative implementation, this can be simplified to

$$\frac{(640320)^{\frac{3}{2}}}{12\pi} = \frac{426880\sqrt{10005}}{\pi} = \sum_{q=0}^{\infty} \frac{(6q)! (545140134q + 13591409)}{(3q)! (q!)^3 (-262537412640768000)^q}$$

There are 3 big integer terms (the multinomial term  $M_q$ , the linear term  $L_q$ , and the exponential term  $X_q$ ) that make up the series, and  $\pi$  equals the constant  $C$  divided by the sum of the series, as below:

$$\pi = C \left( \sum_{q=0}^{\infty} \frac{M_q \cdot L_q}{X_q} \right)^{-1}, \text{ where :}$$

$$C = 426880\sqrt{10005},$$

$$M_q = \frac{(6q)!}{(3q)! (q!)^3},$$

$$L_q = 545140134q + 13591409,$$

$$X_q = (-262537412640768000)^q.$$

The terms  $M_q$ ,  $L_q$ , and  $X_q$  satisfy the following recurrences and can be computed as such:

$$L_{q+1} = L_q + 545140134 \quad \text{where } L_0 = 13591409$$

$$X_{q+1} = X_q \cdot (-262537412640768000) \quad \text{where } X_0 = 1$$

$$M_{q+1} = M_q \cdot \left( \frac{(12q+2)(12q+6)(12q+10)}{(q+1)^3} \right) \quad \text{where } M_0 = 1$$

### 3. Functions & Operations

#### 3.1. Function Prototypes

```
// ---- BigInt functions ----
BigInt new_BigInt(const unsigned int length);
void set_zero(BigInt b);
void free_BigInt(BigInt b);
void print_BigInt(BigInt b);
BigInt Add(const BigInt a, const BigInt b);
BigInt Subtract(const BigInt a, const BigInt b);
void _MUL_(llu x, llu y, llu *carry, llu *result);
BigInt Multiply(const BigInt a, const BigInt b);
void Left_Shift(BigInt num, unsigned int shift);
int Compare(const BigInt a, const BigInt b);
BigInt Divide(const BigInt a, const BigInt b, BigInt *remainder);
char *Decimal_Division(BigInt a, BigInt b);
BigInt Remainder(BigInt a, BigInt b);
BigInt Power(BigInt num, llu p);
BigInt GCD(BigInt a, BigInt b);
BigInt Factorial(llu n);
void precompute_factorial();
void Increment(const BigInt a, const BigInt delta);
void increase_size(BigInt b, const unsigned int delta_len);
void remove_preceding_zeroes(BigInt a);
int isPrime(int n);
int gcd(int a, int b);
```

Figure 1: BIGINT FUNCTION PROTOTYPES.

```
// ---- Complex functions ----
Complex new_Complex();
void print_Complex(Complex a);
void free_Complex(Complex a);
long double real_part(Complex a);
long double imag_part(Complex a);
long double modulus(Complex a);
Complex conjugate(Complex a);
Complex add_Complex(Complex a, Complex b);
Complex subtract_Complex(Complex a, Complex b);
Complex multiply_Complex(Complex a, Complex b);
Complex divide_Complex(Complex a, Complex b);
```

Figure 2: COMPLEX FUNCTION PROTOTYPES.

```

// ---- Fraction functions ----
Fraction new_Fraction();
Fraction input_Fraction();
void print_Fraction(Fraction a);
void reduce_Fraction(Fraction a);
Fraction add_Fraction(Fraction a, Fraction b);
Fraction subtract_Fraction(Fraction a, Fraction b);
Fraction multiply_Fraction(Fraction a, Fraction b);
Fraction divide_Fraction(Fraction a, Fraction b);
void reciprocal_Fraction(Fraction a);
void free_Fraction(Fraction a);
void cancel_zeroes(Fraction a);

```

Figure 3: FRACTION FUNCTION PROTOTYPES.

### 3.2. Operations

#### I. Basic Operations on Big Integers

---

1. Addition : Takes two BIGINT and adds them
2. Subtraction : Takes two BIGINT and subtracts one from the other
3. Multiplication : Takes two BIGINT and multiplies them
4. Division : Takes two BIGINT and divides to give Quotient & Remainder
5. Decimal Division : Takes two BIGINT and divides to give the exact value with decimals
6. Remainder (Modulo) : Takes two BIGINT and gives the remainder
7. GCD : Takes two BIGINT & return GCD of them
8. Power : Takes base(x) as BIGINT & exponent(y) long long int and gives  $x^y$
9. Factorial : Takes a long long int and outputs its factorial

#### II. Operations on Complex Numbers

---

1. Addition : Takes two complex numbers and adds them
2. Subtraction : Takes two complex numbers and subtracts them
3. Multiplication : Takes two complex numbers and multiplies them
4. Division : Takes two complex numbers and divides one from the other
5. Conjugate : Takes a complex number and outputs its conjugate

#### III. Operations on Fractions

---

1. Addition : Takes fractions and adds them
2. Subtraction : Takes two fractions and subtracts them
3. Multiplication : Takes two fractions and multiplies them
4. Division: Takes fractions and divides one from the other
5. Reduce to Simplest Form : Takes a fraction and reduces it to its simplest form.

#### IV. Computation of $\pi$

---

1. Compute Sqrt(10005) using Newton-Raphson Algorithm
2. Compute Value of PI using Chudnovsky Algorithm

#### V. Miscellaneous

---

1. Set Decimal Precision
2. Exit the program

## 4. Output

For the demonstration and showcase of the usability of our program, we are attaching the final output image.

```

└─ TERMINAL
Enter your choice: 23
Enter number of terms of Chudnovsky Algorithm: 215
Computing pi...
Computing term 215... Done!
Calculating SUM of terms...
SUM of terms calculated
Rational Equivalent of pi computed
Execution time: 31.065 seconds
Do you want to convert it to decimal and write it to a file?
Note: Fraction to decimal conversion is very computationally intensive and takes a lot of time.
Your choice? (y/n): y
PI =
3.1415926535897932384626433832795028841971693993751058209749445923078164062862899862803482534211706798214888651328230664789938446095
5058223172535940812848117450284102701938521105559644622948954930819644288109756659334461284756482337867831652712019091456485669234
6934861045426648211393607260249141273724587066666115588174881520920962829254091715364367892590360011330530548204665213841469519415
116094338572703657599195902186117381932611731051854807446217996274956735188375274891227938130119491293307336244656430869213
99463395247371907021798609437027785392171762931767523846748184676694051320095812714526356827785771342577896091736371787214684409
0122495343014654958537105079227968925892354201995611212902196086403441815981362974771309960518707211349999983729780499510597317328
16096318599624459455346908302642523082533446858352619311881710100831783875288658753320838142061717766914730359825349084287554687311
5955286388233787593751957781857708532171226886613001927876611195909216420198938095257801065450532786659361533818279682383019530353
018529689957736259941389124972177528347913115574857242454158695958829533116861727855889075908381754637464939311925508604009277016711
39009848824012858361603563707660104710181942955596198946767837449448255379774726847104047534646208046684259064912933136778289891521
047521628569660240580381501935112533824300355876402474964732639141992726042699227967823547816360093417216412199245863150302861829745
55706749838505404885869269956909272107973693629553211653448872037596023648066549911988184797753566368087426542527862551184175746
7280977727938008616470606104524019217212147723501414197356804816136115735255213475418494048052323987394143134547762416865
189835694855620992192221842725052542568876717904946016534668049886272327917868857843838279679766814541009538837863609506880642251252
051173929848960841284886269456042419652850222106611863067442786220391949450471237137866909563643719172874677646575739624138980658326
459958133904788275900994657640789512694683983525957098258226205224894077267194762684820614769909026401363944374553050682034962524517
49399651431429091908592509372216964615157084838374105978889597729754989301617539284631382086830689427741559185592524593959310409
7524680845987273644695848653836736226260991246088512438843904512441365497627807977156914359977001296160894160486855484063534220
7222582488648158456028506016847294522674676788952521385225499546667278239864565961163548862305774564988355936345681743241125150760
694794518965969940252288797188931456691368672287489405601015033086179286889208747609178249385890697149096759852613655497818931297848
216829084872245800485756401827047551323796414515237462345642850440952638678105114354735709523134272661021550695362314429504
8493718710145765403596279834403742007318578539062186387447888478489683321445713867519435064302184531918484810053706146806749192761
91197939952061419663428754440643745123718192179998391015919561814675142691239748940907186494231961
Execution time: 713.428 seconds
Output Written to file

```

Figure 4: Caption of the Figure.

## 5. Tables

On running the program for different levels of precision of  $\pi$  &  $\sqrt{10005}$ , we are tracking down the average time taken to achieve the required accuracy, and then we are plotting those data in the following table :-

### CALCULATION OF $\pi$

	No. of terms considered(Chudnovsky)	Accuracy in value of $\pi$	Time Taken
1).	10	141	0.002 <i>seconds</i>
2).	70	992	0.53 <i>seconds</i>
3).	210	2977	46.629 <i>seconds</i>
4).	430	6098	894.347 <i>seconds</i>
5).	710	10068	3741.891 <i>seconds</i>

Table 1: Calculation of  $\pi$  & Benchmarks

### CALCULATION OF $\sqrt{10005}$

	No. of terms cons.(Newton-Raphson)	Accuracy in value of $\sqrt{10005}$	Time Taken
1).	1	497	0.018 <i>seconds</i>
2).	3	1995	0.594 <i>seconds</i>
3).	4	3994	3.956 <i>seconds</i>
4).	6	15985	213.331 <i>seconds</i>
5).	7	31971	1682.48 <i>seconds</i>

Table 2: Calculation of  $\sqrt{10005}$  & Benchmarks

## 6. Conclusions

We have been able to store huge numbers that are even out of the range of integers stored in C language. We implemented BIGINT by making various functions to perform all major arithmetic operations. We made complex numbers in C & also implemented Fractions. We constructed algorithms that would take minimum memory and give results in reasonable time complexity. After implementing BIGINT, we used the Newton-Raphson convergence method and Chudnovsky Algorithm to deduce the value of  $\pi$  up to 10000 decimal places.

## 7. Acknowledgements

We would like to thank our Course Instructor (Dr Anil Shukla) and Mentor (Avadhesh Gaur) for providing us with such a significant opportunity to work on this project. We believe that we will indulge in more such projects in the future. We guarantee that this project was created entirely by us and is not a forgery. Finally, We'd like to express our gratitude to our parents and friends for their excellent comments and guidance during the completion of this project.

## 8. References

1. <https://en.wikipedia.org/>
2. <https://cp-algorithms.com/>
3. <https://en.algorithmica.org/>
4. <https://planetmath.org/>

## A. Appendix

1. Detailed information about Newton-Raphson is given in site- <https://en.wikipedia.org/wiki/Newton>
2. Detailed information about Chudnovsky Algorithm is given in site- [https://en.wikipedia.org/wiki/Chudnovsky\\_algorithm](https://en.wikipedia.org/wiki/Chudnovsky_algorithm)