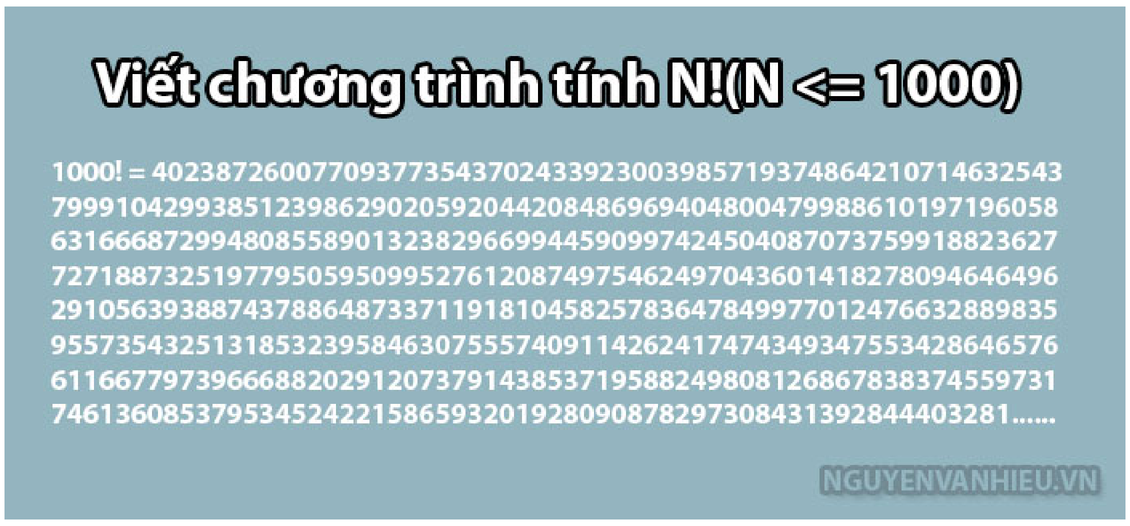
Cộng trừ nhân chia 2 số nguyên lớn trong C/C++

Bởi **Nguyễn Văn Hiếu**

Cộng trừ nhân chia 2 số nguyên lớn trong C/C++ là chủ đề của bài viết này. Nguyễn Văn Hiếu Blog sẽ cung cấp cho các bạn phương pháp để thao tác với số nguyên lớn trong ngôn ngữ lập trình C/C++. Các phép toán cơ bản đối với số nguyên lớn bao gồm: cộng trừ nhân chia 2 số nguyên lớn, và một số phép toán khác nữa.



NỘI DUNG BÀI VIẾT

1. 1. Khi nào cần Bigint trong C/C++?
2. 2. Thao tác với số nguyên lớn trong C/C++
3. 3. Tự code thư viện thao tác với số nguyên lớn
4. 4. Cộng trừ nhân chia 2 số nguyên lớn trong C/C++
    4.1. 4.1. Code cộng hai số nguyên lớn
    4.2. 4.2. Code trừ hai số nguyên lớn
    4.3. 4.3. Một số phép toán với số nguyên lớn khác
    4.4. 4.4. Bài toán tính giai thừa của số lớn
5. Kết luận

1. Khi nào cần Bigint trong C/C++?

Bài toán 1:

Cho hai số nguyên dương A và B (A & B có không quá 1000 chữ số)

Yêu cầu: Tính A + B, A – B, A * B

Các bạn có thể làm thử bài tập này và chấm điểm lời giải của bạn tại đây

Bài toán 2:

Cho số nguyên dương N(N <= 1000). Hãy tính N!

Bàn luận về số nguyên lớn:

Nếu không quan tâm tới phạm vi giá trị của các biến trong 2 bài tập trên. Thì đây là những bài tập cực kỳ đơn giản. Tuy nhiên, nếu bạn code sẽ gặp phải hiện tượng lỗi tràn số.

Nếu bạn chưa biết tràn số là gì? phạm vi giá trị các kiểu dữ liệu là gì, vui lòng đọc trước bài viết này trước khi tiếp tục.

Chúng ta sẽ cần làm việc và quan tâm tới big number khi các kiểu dữ liệu thông thường không thể lưu được những giá trị quá lớn. Khi đó, chúng ta cần xây dựng một chương trình thao tác với số nguyên lớn.

Tôi sẽ cung cấp lời giải 2 ví dụ trên cho bài toán số nguyên lớn của chúng ta ở mục cuối của bài này.

2. Thao tác với số nguyên lớn trong C/C++

Đầu tiên, chúng ta sẽ khai báo cấu trúc struct như sau:

```

const int base = 1000000000; const int base_digits = 9;
struct bigint {
    vector<int> a;
    int sign;

    bigint() :
        sign(1) {
    }
```

8

```
9
10      bigint(long long v) {
11          *this = v;
12      }
13
14      bigint(const string &s) {
15          read(s);
16      }
17  }
18
```

Trong đó, `sign` lưu dấu(số âm, dương) của số nguyên lớn. Mỗi phần tử của vector `a` sẽ lưu một đoạn của số nguyên lớn – đoạn đó nhiều nhất sẽ có 9 chữ số. Như vậy, 1 số nguyên lớn có 90 chữ số chỉ cần vector có size = 10.

Tiếp đó là một số hàm tạo nhận các đối số khác nhau cho kiểu dữ liệu bigint của chúng ta.

Tiếp theo là hàm chuyển string sang số nguyên lớn

```
0
1   void read(const string &s) {
2       sign = 1;
3       a.clear();
4       int pos = 0;
5       while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {
6           if (s[pos] == '-')
7               sign = -sign;
8           ++pos;
9       }
10      for (int i = s.size() - 1; i >= pos; i -= base_digits) {
11          int x = 0;
12          for (int j = max(pos, i - base_digits + 1); j <= i; j++)
13              x = x * 10 + s[j] - '0';
14          a.push_back(x);
15      }
16      trim();
17  }
18
```

Nạp chồng các toán tử nhập và xuất:

```
0
1   friend istream& operator>>(istream &stream, bigint &v) {
2       string s;
3       stream >> s;
4       v.read(s);
5       return stream;
6   }
7
8   friend ostream& operator<<(ostream &stream, const bigint &v) {
9       if (v.sign == -1)
10          stream << '-';
11      stream << (v.a.empty() ? 0 : v.a.back());
12      for (int i = (int) v.a.size() - 2; i >= 0; --i)
13          stream << setw(base_digits) << setfill('0') << v.a[i];
14      return stream;
15  }
16
```

Và nạp chồng một loạt các toán tử khác bao gồm: +, -, *, /, %, +=, -=, *=, /=,...

3. Tự code thư viện thao tác với số nguyên lớn

Dưới đây là full code xử lý số nguyên lớn trong C/C++. Code template này đã include đủ các thư viện cơ bản cần thiết. Việc của bạn là viết thêm hàm main và sử dụng nó.

Do vậy, lời giải ở các phần tiếp theo của bài viết này tôi chỉ đưa ra đoạn code của hàm main. Tôi giả sử rằng bạn đã đưa đoạn code này vào trước hàm main của tôi.

```
0
1   #include <bits/stdc++.h>
2   using namespace std;
3   const int base = 1000000000; const int base_digits = 9;
4   struct bigint {
5       vector<int> a; int sign;
6
7       bigint() :
8           sign(1) {
9       }
10
11      bigint(long long v) {
12          *this = v;
13      }
14
15      bigint(const string &s) {
16          read(s);
17      }
18
19      void operator=(const bigint &v) {
```

8

```
        sign = v.sign;
        a = v.a;
    }

    void operator=(long long v) {
        sign = 1;
        if (v < 0)
            sign = -1, v = -v;
        for (; v > 0; v = v / base)
            a.push_back(v % base);
    }

    bigint operator+(const bigint &v) const {
        if (sign == v.sign) {
            bigint res = v;

            for (int i = 0, carry = 0; i < (int) max(a.size(), v.a.size()) || carry; ++i) {
                if (i == (int) res.a.size())
                    res.a.push_back(0);
                res.a[i] += carry + (i < (int) a.size() ? a[i] : 0);
                carry = res.a[i] >= base;
                if (carry)
                    res.a[i] -= base;
            }
            return res;
        }
        return *this - (-v);
    }

    bigint operator-(const bigint &v) const {
        if (sign == v.sign) {
            if (abs() >= v.abs()) {
                bigint res = *this;
                for (int i = 0, carry = 0; i < (int) v.a.size() || carry; ++i) {
                    res.a[i] -= carry + (i < (int) v.a.size() ? v.a[i] : 0);
                    carry = res.a[i] < 0;
                    if (carry)
                        res.a[i] += base;
                }
                res.trim();
                return res;
            }
            return -(v - *this);
        }
        return *this + (-v);
    }

    void operator*=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = 0, carry = 0; i < (int) a.size() || carry; ++i) {
            if (i == (int) a.size())
                a.push_back(0);
            long long cur = a[i] * (long long) v + carry;
            carry = (int) (cur / base);
            a[i] = (int) (cur % base);
            //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A"(cur), "c"(base));
        }
        trim();
    }

    bigint operator*(int v) const {
        bigint res = *this;
        res *= v;
        return res;
    }

    friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1) {
        int norm = base / (b1.a.back() + 1);
        bigint a = a1.abs() * norm;
        bigint b = b1.abs() * norm;
        bigint q, r;
        q.a.resize(a.a.size());

        for (int i = a.a.size() - 1; i >= 0; i--) {
            r *= base;
            r += a.a[i];
            int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
            int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
            int d = ((long long) base * s1 + s2) / b.a.back();
            r -= b * d;
            while (r < 0)
                r += b, --d;
            q.a[i] = d;
        }

        q.sign = a1.sign * b1.sign;
        r.sign = a1.sign;
        q.trim();
        r.trim();
        return make_pair(q, r / norm);
    }

    bigint operator/(const bigint &v) const {
```

8

```
        return divmod(*this, v).first;
    }

    bigint operator%(const bigint &v) const {
        return divmod(*this, v).second;
    }

    void operator/=(int v) {
        if (v < 0)
            sign = -sign, v = -v;
        for (int i = (int) a.size() - 1, rem = 0; i >= 0; --i) {
            long long cur = a[i] + rem * (long long) base;
            a[i] = (int) (cur / v);
            rem = (int) (cur % v);
        }
        trim();
    }

    bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
    }

    int operator%(int v) const {
        if (v < 0)
            v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
            m = (a[i] + m * (long long) base) % v;
        return m * sign;
    }

    void operator+=(const bigint &v) {
        *this = *this + v;
    }
    void operator-=(const bigint &v) {
        *this = *this - v;
    }
    void operator*=(const bigint &v) {
        *this = *this * v;
    }
    void operator/=(const bigint &v) {
        *this = *this / v;
    }

    bool operator<(const bigint &v) const {
        if (sign != v.sign)
            return sign < v.sign;
        if (a.size() != v.a.size())
            return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i])
                return a[i] * sign < v.a[i] * sign;
        return false;
    }

    bool operator>(const bigint &v) const {
        return v < *this;
    }
    bool operator<=(const bigint &v) const {
        return !(v < *this);
    }
    bool operator>=(const bigint &v) const {
        return !(*this < v);
    }
    bool operator==(const bigint &v) const {
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const bigint &v) const {
        return *this < v || v < *this;
    }

    void trim() {
        while (!a.empty() && !a.back())
            a.pop_back();
        if (a.empty())
            sign = 1;
    }

    bool isZero() const {
        return a.empty() || (a.size() == 1 && !a[0]);
    }

    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }

    bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
```

8

```
    }

    long long longValue() const {
        long long res = 0;
        for (int i = a.size() - 1; i >= 0; i--)
            res = res * base + a[i];
        return res * sign;
    }

    friend bigint gcd(const bigint &a, const bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const bigint &b) {
        return a / gcd(a, b) * b;
    }

    void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int) s.size() && (s[pos] == '-' || s[pos] == '+')) {
            if (s[pos] == '-')
                sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }

    friend istream& operator>>(istream &stream, bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }

    friend ostream& operator<<(ostream &stream, const bigint &v) {
        if (v.sign == -1)
            stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int) v.a.size() - 2; i >= 0; --i)
            stream << setw(base_digits) << setfill('0') << v.a[i];
        return stream;
    }

    static vector<int> convert_base(const vector<int> &a, int old_digits, int new_digits) {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
        for (int i = 1; i < (int) p.size(); i++)
            p[i] = p[i - 1] * 10;
        vector<int> res;
        long long cur = 0;
        int cur_digits = 0;
        for (int i = 0; i < (int) a.size(); i++) {
            cur += a[i] * p[cur_digits];
            cur_digits += old_digits;
            while (cur_digits >= new_digits) {
                res.push_back(int(cur % p[new_digits]));
                cur /= p[new_digits];
                cur_digits -= new_digits;
            }
        }
        res.push_back((int) cur);
        while (!res.empty() && !res.back())
            res.pop_back();
        return res;
    }

    typedef vector<long long> vll;

    static vll karatsubaMultiply(const vll &a, const vll &b) {
        int n = a.size();
        vll res(n + n);
        if (n <= 32) {
            for (int i = 0; i < n; i++)
                for (int j = 0; j < n; j++)
                    res[i + j] += a[i] * b[j];
            return res;
        }

        int k = n >> 1;
        vll a1(a.begin(), a.begin() + k);
        vll a2(a.begin() + k, a.end());
        vll b1(b.begin(), b.begin() + k);
        vll b2(b.begin() + k, b.end());

        vll a1b1 = karatsubaMultiply(a1, b1);
        vll a2b2 = karatsubaMultiply(a2, b2);

```

8

```
        for (int i = 0; i < k; i++)
            a2[i] += a1[i];
        for (int i = 0; i < k; i++)
            b2[i] += b1[i];

        vll r = karatsubaMultiply(a2, b2);
        for (int i = 0; i < (int) a1b1.size(); i++)
            r[i] -= a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
            r[i] -= a2b2[i];

        for (int i = 0; i < (int) r.size(); i++)
            res[i + k] += r[i];
        for (int i = 0; i < (int) a1b1.size(); i++)
            res[i] += a1b1[i];
        for (int i = 0; i < (int) a2b2.size(); i++)
            res[i + n] += a2b2[i];
        return res;
    }

    bigint operator*(const bigint &v) const {
        vector<int> a6 = convert_base(this->a, base_digits, 6);
        vector<int> b6 = convert_base(v.a, base_digits, 6);
        vll a(a6.begin(), a6.end());
        vll b(b6.begin(), b6.end());
        while (a.size() < b.size())
            a.push_back(0);
        while (b.size() < a.size())
            b.push_back(0);
        while (a.size() & (a.size() - 1))
            a.push_back(0), b.push_back(0);
        vll c = karatsubaMultiply(a, b);
        bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int) c.size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int) (cur % 1000000));
            carry = (int) (cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits);
        res.trim();
        return res;
    }
};
```

4. Cộng trừ nhân chia 2 số nguyên lớn trong C/C++

4.1. Code cộng hai số nguyên lớn

Ở đoạn code template phía trên, tôi đã có ghi đè các toán tử `+` , `+=` , `>>` và `<<` . Do vậy, để cộng hai số nguyên lớn, ta làm đơn giản như sau:

```
int main(){
    bigint n1, n2;
    cout << "\nNhap so thu nhat: ";
    cin >> n1;
    cout << "\nNhap so thu hai : ";
    cin >> n2;
    cout << "Tong 2 so = " << (n1 + n2) << '\n';
}
```

Kết quả:

```
Nhap so thu nhat: 9999999999
Nhap so thu hai : -1111111111

Tong 2 so = 8888888888
```

4.2. Code trừ hai số nguyên lớn

Giống như phép toán cộng số nguyên lớn phía trên, phép trừ ta làm tương tự.

```
int main(){
    bigint n1, n2;
    cout << "\nNhap so thu nhat: ";
    cin >> n1;
    cout << "\nNhap so thu hai : ";
    cin >> n2;
    cout << "Tong 2 so = " << (n1 - n2) << '\n';
}
```

8

4.3. Một số phép toán với số nguyên lớn khác

Phép nhân chia 2 số nguyên lớn

Bạn có thể làm tương tự như phép cộng và trừ tôi đã làm ở trên. Còn dưới đây tôi sẽ thử sử dụng hàm tạo bigint từ biến string.

```
0
1  int main(){
2      string s1 = "12345", s2 = "-5";
3      bigint n1(s1), n2(s2);
4      cout << s1 << " * " << s2 << " = " << (n1*n2) << '\n';
5      cout << s1 << " / " << s2 << " = " << (n1/n2) << '\n';
6      // Hoặc sử dụng toán tử '*='
7      // Sử dụng các toán tử +=, -=, /=, %= tương tự
8      n1 *= n2;
9      cout << s1 << " * " << s2 << " = " << n1 << '\n';
10 }
11
```

Kết quả:

```
0
1 12345 * -5 = -61725
2 12345 / -5 = -2469
3 12345 * -5 = -61725
4
```

Tìm UCLN, BCNN của 2 số nguyên lớn

```
0
1  int main(){
2      bigint n1, n2;
3      cout << "\nNhap so thu nhat: ";
4      cin >> n1;
5      cout << "\nNhap so thu hai : ";
6      cin >> n2;
7      cout << "UCLN: " << gcd(n1,n2) << '\n';
8      cout << "BCNN: " << lcm(n1,n2) << '\n';
9  }
10
```

Kết quả:

```
0
1 Nhap so thu nhat: 9999
2
3 Nhap so thu hai : 111
4 UCLN: 3
5 BCNN: 369963
6
```

Ngoài ra, còn rất nhiều toán tử khác giúp chúng ta làm việc với số nguyên lớn. Ở đây tôi chỉ trình bày cộng trừ nhân chia 2 số nguyên lớn. Các bạn có thể tìm hiểu và sử dụng các hàm, toán tử khác có sẵn trong code template trên.

4.4. Bài toán tính giai thừa của số lớn

Với bài toán số 1, chính là bài toán cộng trừ nhân chia 2 số nguyên lớn. Và tôi đã giải quyết nó ở phía trên. Sau đây, chúng ta sẽ sử dụng template bigint trong c++ phía trên để tính giai thừa số nguyên lớn nhé.

Để tính giai thừa của số nguyên lớn, không có cách nào khác là nhân các phần tử từ 1,2,3,…,n với nhau. Tích của chúng sẽ là lời giải của bài toán này.

Với bài toán tính giai thừa của số nguyên lớn N(N <= 1000). Do đó, ta vẫn nhập N là số kiểu `int` nhé.

Sử dụng code template tôi cung cấp phía trên, và đây là hàm main chúng ta cần phải viết:

```
0
1  int main(){
2      bigint answer = 1;
3      int N;
4      cout << "Nhap so N = ";
5      cin >> N;
6      // Tinh giai thua
7      for(int i = 2; i <= N; ++i){
8          answer *= i;
9      }
10     cout << N <<"! = " << answer << '\n';
11 }
12
```

Và đây là ảnh chụp kết quả của phép tính 1000!

Kết luận

Như vậy, Nguyễn Văn Hiếu vừa cùng các bạn đi giải quyết hầu hết các bài toán cần xử lý với số nguyên lớn trong C/C++. Giờ đây, việc cộng trừ nhân chia 2 số nguyên lớn đã trở nên đơn giản hơn bao giờ hết khi chúng ta đã nạp chồng toán tử cho cấu trúc bigint. Hi vọng bài viết mang lại nhiều kiến thức bổ ích cho các bạn độc giả!

Chúc các bạn học tập tốt!

Nguyễn Văn Hiếu

Sáng lập cộng đồng Lập Trình Không Khó với mong muốn giúp đỡ các bạn trẻ trên con đường trở thành những lập trình viên tương lai. Tất cả những gì tôi viết ra đây chỉ đơn giản là sở thích ghi lại các kiến thức mà tôi tích lũy được.

Lập Trình Không Khó là một cộng đồng chia sẻ và đào tạo lập trình phi lợi nhuận hàng đầu tại Việt Nam.

Liên hệ hợp tác & quảng cáo

BÀI VIẾT HAY

Bài 1. Giới thiệu khóa học “Học C Bá Đạo”
21/07/2019

Bài 45. Sắp xếp dãy số giảm dần, tăng dần
01/09/2019

Danh sách liên kết đơn – Single linked list
13/01/2020

CHUYÊN MỤC HAY

Học C/C++ 191
Học Python 46
Học Java 45
Học Javascript 36
Bài tập Javascript 23
Học Web 23
Machine learning 21
Chia sẻ 17

- BẠN BÈ & ĐỐI TÁC -

Tự Học Đồ Họa Cách Học Lập Trình VNTALKING

© 2018-2020. Bản quyền thuộc Lập Trình Không Khó. Privacy & Terms

8