CS205 · C/C++ Programming

Project 1 - Calculator

何泽安 (He Zean)
12011323
Sep. 9, 2021

Contents

1 Analysis

The very first problem that needs to be solved to implement a calculator is to accept data. To control whether or not to accept command line arguments, the most intuitive way (which is also mentioned in the course) is to receive argc and argv in the main function and check the value of *argc*. Considering the naive way of using int to receive the input steam of cin may cause several problems and thus makes our program less robustic, and noting that *argv* is a C-style string, we consider converting it to string and waiting for the next step. In addition, considering that the 「integer」 in the document is ambiguous, that it may refer to an infinitely long mathematically significant integer, it is even more necessary to use an unlimited-length data type like string to receive it. Next, we deal with the data. If the int in the document refers to int (Version 1 below), we need to prevent overflow, if it is an infinitely long integer, we need to use other techniques to implement the multiplication of string or to wrap it into a better data structure.

1.1 Handling different input methods

As Requirement 1 & 2 shows, our program should be able to recognize whether the user input two multipliers by the command line arguments or not. Three possible conditions should be covered:

① the user runs the program without specifying any arguments other than *./mul* ;
② the user runs the program, with two integers given by the command line arguments ;
③ the user gives more or less then two integers in the command line arguments [1].

1.2 Input validity checking

Noticing that some non-integer may be input, the program should be able to check if the input only contains numbers (say, from 0 to 9) and the negative sign. But that's not enough, we shall also make sure that the two inputs don't *overflow* the integer limit [2] before performing the next step.

1.3 Multiplying large integers

I consider that 「integer」 repersents 「int」 in C++ in the preceding parts, but after I finishing one version, I realized this ambiguity, thus two versions of program are kept.

1.3.1 Version 1 - 「int」 s

As we know, in most x86/x64 platforms, one int takes 4 bytes and theus can store number within -2^{31} to $2^{31}-1$, multiplying two integers may produce up to about 2^{62}, which exceeds the limit of integer. While one *long long* takes 8 bytes [3] (in x86/x64 platforms), it can safely contain the product.

1.3.2 Version 2 - infinite length 「integer」

Though it's said "infinite length", typically we won't handle that large number, we can thus let the max length of our program as a fixed number (here we define it in a macros for easier adaption). As I mentioned in Section 4.3, we can use int or char arrays to store extra large numbers, but here I will use a *struct big_num* to store them.

*To make the use of this data structure more concise and elegant, I have override its operator +, - and big_num **

[1]Since the document doesn't specify what response is expected, I prefer that when only one integer is received, the program will ask the user to input another one, when more then two are received, the program will take the first two and warn the user.

[2]Though the product of zero and any number is still zero, we still insist this, as the document required.

[3]long long - target type will have width of at least 64 bits. (since C++11, see the offical document)

ordinary int / long, which also means that only by adjusting one line of code in the main function can we make our calculator support addition and subtraction of very long integers.

1.4 Features in this project

① **User friendly Interaction** As Section 1.1 mentioned, when user input one or more than three numbers by command line argument, the program can warn the user.

② **Check the input by regular expression** By declaring a regex pattern, can we easily check the input's form. Please also notice that I wrote three versions of input checker, since <regex> is not in C++11 standard, there's also a function that check the input char by char. Also, I've considered the positive sign.

③ **Retry until correct** Rather than exit the program directly, my program allow the user to re-enter two number until two valid integers are input or manually exit the program.

④ **Big number support** See the code in Section 2.2. Please notice that in my source.cpp, you need to replace the main function by the code in this report (actually, the differences are just in a few lines) to enable this feature.

2 Code

2.1 Version 1

2.1.1 Input checker

```
#include <string>
#include <regex>

using namespace std;

const regex pattern("^[+-]?[0-9]*$");  // an integer should at least looks like this

/**
 * Check if the input is a valid int
 *
 * @param s (maybe) int-like string
 * @return the integer notated by s
 * @throws invalid_argument if the input is not purely numeric
 * @throws out_of_range if the input exceeds the range of int
 */
int input_checker(const string &s) {
    if (!regex_match(s, pattern)) {
        throw invalid_argument("Input is not an integer");
    }
    try {
        return stoi(s);
    } catch (out_of_range &e) {
        /*
         * though stoi may mainly throws exception
         * <out_of_range> and <invalid_argument>, we have
         * used regex to make sure that any string passed to
         * this function is number-like, thus it's only possible
         * for stoi to throw <out_of_range>
         */
        throw e;
    }
}
```

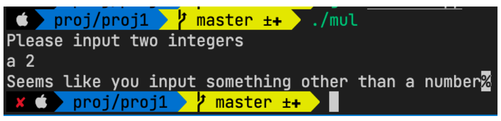
Considering that *regex* is an unapproved C++ 11 header, we can use the following method to check the whole input manually, though regex is more elegent.

```
int input_checker_2(const string &s) {
    if (!(s[0] == '+' || s[0] == '-' || (s[0] >= '0' && s[0] <= '9'))) {
        throw invalid_argument("Input is not an integer");
    }
    for (int i = 0; i < s.length(); i++) {
        if (s[0] > '9' || s[i] < '0') {
            throw invalid_argument("Input is not an integer");
        }
    }
    try {
        return stoi(s);
    } catch (out_of_range &e) {
        throw e;
    }
}
```

2.1.2 Input method selector

```
#include <iostream>
/* Codes in code block 2.1.1 */
int main(int argc, char **argv) {
    int x, y;
    string temp;
    long long product;
    try {
        if (argc == 1) {  // user should input two numbers now
            cout << "Please input two integers" << endl;
            cin >> temp;
            x = input_checker(temp);
            cin >> temp;
            y = input_checker(temp);
        } else if (argc == 2) {  // one number is passed by command line args,
                                 // another one is now to be input
            x = input_checker(string(argv[1]));
17              // If x is invalid, the program will quit now,
18              // otherwise it will ask the user to input another one
19              cout << "You input " << x
20                   << " by command line args, please input another integer"
21                   << endl;
22              cin >> temp;
23              y = input_checker(temp);
24          } else {  // there are two or more inputs passed by command line
25              x = input_checker(string(argv[1]));
26              y = input_checker(string(argv[2]));
27              if (argc > 3) {
28                  cerr << "You input more arguments than expected, the rest ones are ignored"
29                       << endl;
30              }
31          }
32      } catch (invalid_argument &) {
33          cerr << "Seems like you input something other than a number";
34          return -1;
35      } catch (out_of_range &) {
36          cerr << "The number you input is too large or small that exceeds "
37                  "the range of int";
38          return -1;
39      }
40
41      // Upon here we have two valid ints
42      /* Codes */
43  }
```

Consider the above code *line 34* in the try block, we can adopt our code to enable the user to continue inputting until two valid integers are accepted. The above code performs like this.

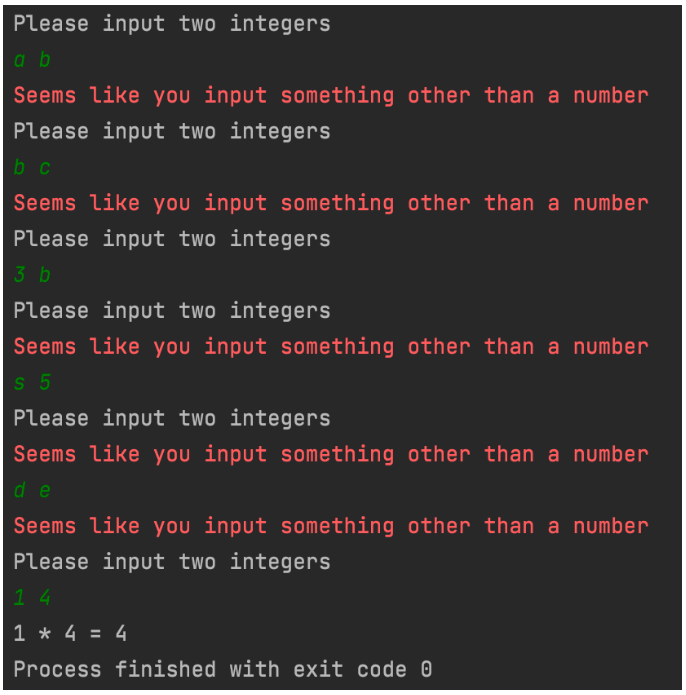


```
 proj/proj1  master ±+  ./mul
Please input two integers
a 2
Seems like you input something other than a number%
 ✘  proj/proj1  master ±+ 
```

```
bool has_invalid = false;

do {
    try {
    /* Codes */
    } catch (invalid_argument &) {
        cerr << "Seems like you input something other than a number\n";
        argc = 1;  // it forces the program to go into the first if block and receive two nums
        has_invalid = true;
        cin.ignore(INT_MAX,'\n');
    } catch (out_of_range &) {
        cerr << "The number you input is too large or small that exceeds "
                "the range of int\n";
        argc = 1;
```

```
            has_invalid = true;
            cin.ignore(INT_MAX,'\n');
        }
    } while (has_invalid);
```



```
Please input two integers
a b
Seems like you input something other than a number
Please input two integers
b c
Seems like you input something other than a number
Please input two integers
3 b
Please input two integers
Seems like you input something other than a number
s 5
Please input two integers
Seems like you input something other than a number
d e
Seems like you input something other than a number
Please input two integers
1 4
1 * 4 = 4
Process finished with exit code 0
```

2.1.3 Calculator

```
/* Codes in code block 2.1 */
int main(int argc, char **argv) {
    int x, y;
    string temp;
    long long product;
    /* Codes in code block 2.2 provided two ints x and y*/
    product = static_cast<long long>(x) * y;
    cout << x << " * " << y << " = " << product;
    return 0;
}
```

2.2 Version 2

```
#include <iostream>
#include <string>

#define MAX_SIZE 100

using namespace std;

// Defining struct big_num and overriding its operators
typedef struct big_num {
```

```cpp
private:
    int *d_;
    int len_;
    bool positive_;

public:
    big_num() {
        d_ = new int[MAX_SIZE];
        len_ = 0;
        positive_ = true;
    }

    ~big_num() {
        delete[] d_;
        d_ = nullptr;
    }

    big_num bias(int bias) {
        big_num temp;
        memset(temp.d_, 0, sizeof(int) * bias);
        memcpy(temp.d_ + bias, this->d_, static_cast<int>(sizeof(int)) * this->len_);
        temp.len_ = this->len_ + bias;
        temp.positive_ = this->positive_;
        return temp;
    }

    /**
     * @param str the string has passed the check by <regex pattern>
     */
    big_num &operator=(const string &str) {
        bool has_flag = str[0] == '-' || str[0] == '+';
        int str_len = has_flag ? str.length() - 1 : str.length();
        this->len_ = 0;
        this->positive_ = str[0] != '-';
        for (int i = 0; i < str.length(); i++)
            if (str[str_len - this->len_ - 1] != '0') {
                this->len_++;
                this->d_[this->len_] = str[str_len - this->len_ - 1] - '0';
        }
        return *this;
    }

    big_num operator+(const big_num &op) {
        big_num sum;
        int temp, carry = 0;
        for (int i = 0; i < this->len_ || i < op.len_; i++) {
            temp = this->d_[i] + op.d_[i] + carry;
            sum.d_[sum.len_++] = temp % 10;
            carry = temp / 10;
        }
        if (carry)
            sum.d_[sum.len_++] = carry;
        return sum;
    }

    big_num operator*(int64_t op) {
        big_num ans;
        ans.positive_ = this->positive_ ^ (op < 0);
        int temp, carry = 0;
        for (int i = 0; i < this->len_; i++) {
            temp = this->d_[i] * op + carry;
            ans.d_[ans.len_++] = temp % 10;
            carry = temp / 10;
        }
        while (carry) {
            ans.d_[ans.len_++] = carry % 10;
            carry = carry / 10;
        }
        return ans;
    }

    big_num operator*(big_num &op) {
        big_num prod;
        prod = "0";
        // prevent overflow
        if (this->len_ + op.len_ > MAX_SIZE) {
            delete[] prod.d_;
            prod.d_ = nullptr;
            prod.d_ = new int[2 * MAX_SIZE];
        }
        for (int i = 0; i < this->len_; i++)
            prod = prod + (op * this->d_[i]).bias(i);
        return prod;
    }

    string show() {
        string s;
        if (!this->positive_) cout << "-";
        for (int i = this->len_ - 1; i >= 0; i--)
            s.push_back(this->d_[i] + '0');
        return s;
    }
} big_num;
```

```
big_num input_checker_b(const string &s) {
    if (!regex_match(s, pattern))
        throw std::invalid_argument("Input is not an integer");
    big_num temp;
    temp = s;
    return temp;
}
```

```
int main(int argc, char **argv) {
    big_num x, y;
    string temp;
    try {
        if (argc == 1) {  // user should input two numbers now
            cout << "Please input two integers" << endl;
            cin >> temp;
            x = input_checker_b(temp);
            cin >> temp;
            y = input_checker_b(temp);
        } else if (argc == 2) {  // one number is passed by command line args,
                                 // one to be input
            x = input_checker_b(string(argv[1]));
            // If x is invalid, the program will quit now, otherwise it will ask
            // the user to input another one
            cout << "You input " << x
                 << " by command line args, please input another integer"
                 << endl;
            cin >> temp;
            y = input_checker_b(temp);
        } else {  // there are two or more inputs passed by command line
            x = input_checker(string(argv[1]));
            y = input_checker(string(argv[2]));
            if (argc > 3) {
                cerr << "You input more arguments than expected, the rest ones are ignored"
                     << endl;
            }
        }
    } catch (invalid_argument &) {
        cerr << "Seems like you input something other than a number";
        return -1;
    }

    big_num product = x * y;  // operator* is overrided
    cout << x.show() << " * " << y.show() << " = " << product.show();
    return 0;
}
```

3 Result & Verification

```
$ cd /path/to/project/ && g++ source.cpp -std=c++20 -o mul
```

Test case #1: Normal case

```
$ ./mul
[Output] Please input two integers
[ Input] 2 3
[Output] 2 * 3 = 6
```

Test case #2: Pass 2 ints by command line args

```
$ ./mul 2 3
[Output] 2 * 3 = 6
```

Test case #3: Pass 1 int by command line args

```
$ ./mul 2
[Output] You input 2 by command line args, please input another integer
[ Input] 3
[Output] 2 * 3 = 6
```

Test case #4: Pass too many command line args

```
$ ./mul 2 3 4 5
[Output] You input more arguments than expected, the rest ones are ignored
[Output] 2 * 3 = 6
```

Test case #5: Input non-integer numbers - 1

```
$ ./mul
[Output] Please input two integers
[ Input] a 2
[Output] Seems like you input something other than a number
[Process finished with exit code 255]
```

Test case #6: Input non-integer numbers - 2 (float number)

```
$ ./mul
[Output] Please input two integers
[ Input] 1.1 2.3
[Output] Seems like you input something other than a number
[Process finished with exit code 255]
```

Version 1

Test case #7.1: Input non-integer numbers - 3 (int overflow)

```
$ ./mul
[Output] Please input two integers
[ Input] 100000000000 12
[Output] The number you input is too large or small that exceeds the range of int
[Process finished with exit code 255]
```

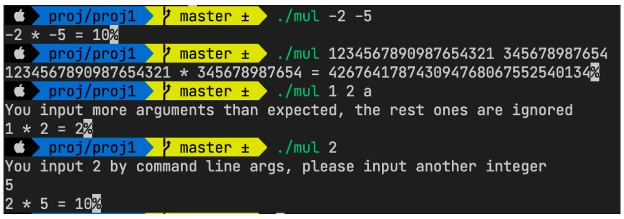
Test case #7.2: Big int multiplication

```
$ ./mul
[Output] Please input two integers
[ Input] 1234567890 1234567890
[Output] 1234567890 * 1234567890 = 1524157875019052100
```

Version 2

Test case #8: Big integer that exceeds int limit

```
$ ./mul
[Output] Please input two integers
[ Input] 12345678987654321 12345678912345678
[Output] 152415788736473088736473122374638
// Take it easy, I've checked it by Mathematica, that's right
```



```
 proj/proj1  master ±  ./mul -2 -5
-2 * -5 = 10%
 proj/proj1  master ±  ./mul 1234567890987654321 345678987654
1234567890987654321 * 345678987654 = 426764178743094768067552540134%
 proj/proj1  master ±  ./mul 1 2 a
You input more arguments than expected, the rest ones are ignored
1 * 2 = 2%
 proj/proj1  master ±  ./mul 2
You input 2 by command line args, please input another integer
5
2 * 5 = 10%
```

4 Difficulties & Solutions

4.1 Converting string / char[] into integer

This part is for solving version 1, version 2 just check whether the input is a number and directly store it.

When trying to use *atoi(const char *str)*, IDE suggestes us to use *strtol* instead. Also, I found that *atoi(const char *str)* cannot handle int overflow, and don't report errors when converting improper chars.

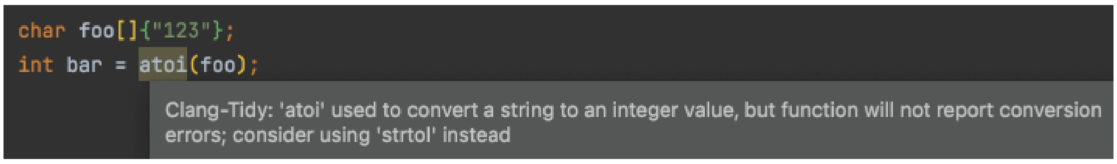
Solution Function *stoi(const std::string& str)* is easy to use, since *string* is more recommended than the C-style string and it can throw exception when convertion error happens. However, if a string starts with number 0 to 9 or the string contains dots, stoi doesn't report. We can use *Reuglar Expression* to check this [4]. Of course check the string char by char (the first char may be '+' '-' or a number and the left ones are all numbers '0' to '9') is another typical way, but regex is more elegent.

Another way is to use *strtol*, but I'm too tired to write about it here.

4.2 Clearing the cin buffer

About the discussion in section 2.1.2, at first I search solutions in bing and baidu, as they instructed I wrote the following code, however, this could not clean all the buffer and get ready for the next turn as I excepted. It performs as the figure shows.

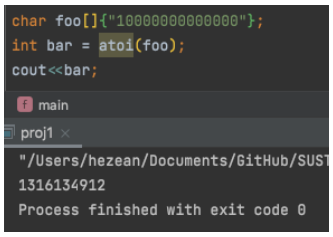
[4] If it passes the regex check, at least it looks like a integer, but may overflow, we then use stoi to prevent it.



```
char foo[]{"123"};
int bar = atoi(foo);
```

Clang-Tidy: 'atoi' used to convert a string to an integer value, but function will not report conversion errors; consider using 'strtol' instead

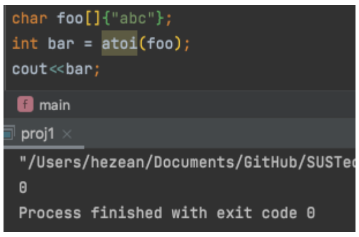
(a) 'atoi' used to convert a string to an integer value, but function will not report conversion errors



```
char foo[]{"10000000000000"};
int bar = atoi(foo);
cout<<bar;
```

```
"/Users/hezean/Documents/GitHub/SUST
1316134912
Process finished with exit code 0
```

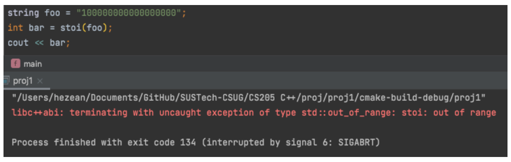
(b) If the char array represents a number exceeding the int's limit, atoi will return a wrong number



```
char foo[]{"abc"};
int bar = atoi(foo);
cout<<bar;
```

```
"/Users/hezean/Documents/GitHub/SUSTe
0
Process finished with exit code 0
```

(c) When the argument is not a integer-like char array, function atoi just returns 0

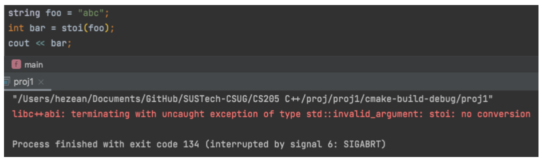


```
string foo = "100000000000000000";
int bar = stoi(foo);
cout << bar;
```

```
"/Users/hezean/Documents/GitHub/SUSTech-CSUG/CS205 C++/proj/proj1/cmake-build-debug/proj1"
libc++abi: terminating with uncaught exception of type std::out_of_range: stoi: out of range

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

(d) stoi reports int overflow



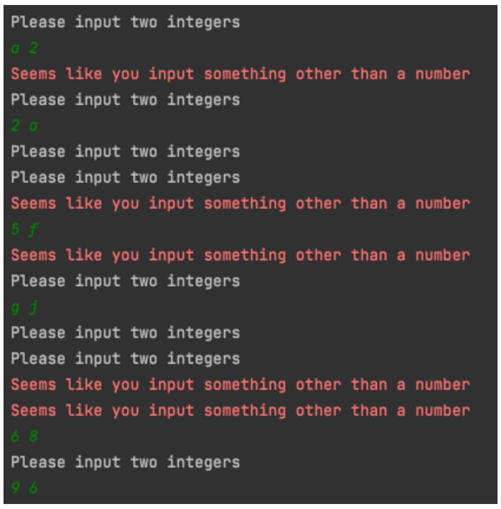
```
string foo = "abc";
int bar = stoi(foo);
cout << bar;
```

```
"/Users/hezean/Documents/GitHub/SUSTech-CSUG/CS205 C++/proj/proj1/cmake-build-debug/proj1"
libc++abi: terminating with uncaught exception of type std::invalid_argument: stoi: no conversion

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

(e) stoi reports improper input

```cpp
try {
/* Codes */
} catch (invalid_argument &) {
    cerr << "Seems like you input something other than a number\n";
    argc = 1;  // it forces the program to go into the first if block and receive two nums
    has_invalid = true;
    cin.sync();
}
/* Codes */
} while (has_invalid);
```



```
Please input two integers
a 2
Seems like you input something other than a number
Please input two integers
2 a
Please input two integers
Please input two integers
Seems like you input something other than a number
5 f
Seems like you input something other than a number
Please input two integers
g j
Please input two integers
Please input two integers
Seems like you input something other than a number
Seems like you input something other than a number
6 8
Please input two integers
9 6
```

Finally I found a proper solution on StackOverflow (see: `https://stackoverflow.com/questions/23713346/clear-the-cin-buffer-before-another-input-request-c`). It uses *std::numeric_limits<std::strea* to represent the biggest length of cin's buffer, in my project, I used INT_MAX instead. It is suggested that using the combination of cin.ignore() and cin.clear() to clear the buffer and reset the status, but since we used string to receive the input, we therefore need not cin.clear().

4.3 Exception Handling in C++

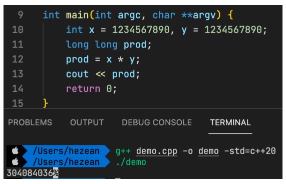
My input checker may throws exceptions, then the main function will then catch it, and print some friendly warnings. At first I didn't know how to throw an user-defined exception (Code 2.1, Line 17), by referring to the book and searching the Internet [5], I learnt how to write it.

Please notice that when I was programming, I consider that the calculator only need to computer once, and any incorrect input should lead to a warning and then quit. Actually, there are several ways to re-run the program, like using flags, *goto statements (through is not recommended)*, etc.

4.4 Explicit Cast

This part is only for solving version 1.

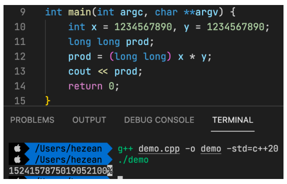
The following figure shows that multiplying two ints without explicit casting (one or both of them) the int into long long, the RHS uses int at first, then implicit case the (wrong) product into long long, it doesn't work as we want. This feature just like Java, and can be simply solved by Fig (b), according to Google C++ Style Guide, using *static_cast<long long>()* is better in format than using the C style casting.



```
int main(int argc, char **argv) {
    int x = 1234567890, y = 1234567890;
    long long prod;
    prod = x * y;
    cout << prod;
    return 0;
}
```

```
/Users/hezean  g++ demo.cpp -o demo -std=c++20
/Users/hezean  ./demo
304084036%
```

(a) RHS first overflows, then be cast into long long



```
int main(int argc, char **argv) {
    int x = 1234567890, y = 1234567890;
    long long prod;
    prod = (long long) x * y;
    cout << prod;
    return 0;
}
```

```
/Users/hezean  g++ demo.cpp -o demo -std=c++20
/Users/hezean  ./demo
1524157875019052100%
```

(b) Now RHS is long long, and then be passed to LHS

4.5 Defining struct big_num

Ideally this should be defined as a class and be seperated to header file and source file, but since we only submit one file, it is simplified to a structure. As I only wrote classes in Java before, there are numerous difference between C++ and Java that I need to notice. For example, the usage of reference and pointers, constructor and deconstructor and overloadding operators. To overcome these difficulties, I read relevant chapters of C++ Primer Plus and Effective C++.

There's another way to store big integers like using string, but this leads to a manual management during operating, but this can make the code less elegant. Defining a structure can capsulize these tedious details and take full advantage of the object-oriented features supported by C++. Since I override operator $big\_num + big\_num$ and $big\_num * int$, things get easier since a *vertical multiplication* is a set (sum) of multiplications between big_num and int.

[5]One of useful reference `https://blog.csdn.net/wbentely/article/details/70240526`

```
// before: using string to apply big_num multiplication
// since I didn't use this method in my project, the following code is extracted entirely from cnblogs
// reference: \url{https://www.cnblogs.com/alderheart/p/10958615.html}
string multiply(string num1, string num2) {
    int l1 = num1.length(), l2 = num2.length();
    if (num1 == "0" || num2 == "0") return "0";
    string res(l1 + l2, '0');
    for (int i = l1 - 1; i >= 0; i--) {
        int step = 0;
        for (int j = l2 - 1; j >= 0; j--) {
            int mul = (num1[i] - '0') * (num2[j] - '0');
            int sum = res[i + j + 1] - '0' + step + mul % 10;
            res[i + j + 1] = sum % 10 + '0';
            step = sum / 10 + mul / 10;
        }
        res[i] += step;
    }
    for (int i = 0; i < l1 + l2; i++)
        if (res[i] != '0') return res.substr(i);
    return "0";
}

// after: overriding operators can make other operations easier
// though in this project the code of defining struct may be longer than the above way
// this code is much more flexible and can be easily adapted to other usages
// and can support senior operations
big_num operator*(big_num &op) {
    for (int i = 0; i < this->len_; i++)
        prod = prod + (op * this->d_[i]).bias(i);
//                  ^     ^~~ (big_num * int) -> big_num
//                  |~~ (big_num + big_num) -> big_num
    return prod;
}
```

4.6 Better than Vertical Multiplication: Karatsuba Multiplication

The most part of this algorithm[6] *comes from Stanford Lagunita Algorithms by Tim Roughgarden. I consider that 「integer」 repersents 「int」 in C++ in the preceding parts, but after I finishing one version, I realized this ambiguity, thus two versions of program are kept.* Its time complexity is $O(n^{\log 3})$. In code block 2.1.2 I analogged vertical multiplication instead of using this, however.

```
procedure Karatsuba(num1,num2)
    if (num1 < 10) or (num2 < 10) then
        return num1 * num2

    // Calculates the size of the numbers
    m = max(size(num1), size(num2))
    m2 = m / 2

    high1, low1 = split_at(num1, m2)
```

[6]See my blog `https://hezean.github.io/2021/07/19/dsaa-stanford-1.3/` , code was written by me in this summer vacation, with refering somem references that are listed there.

```
    high2, low2 = split_at(num2, m2)

    // Calls made to numbers approximately half the size
    z0 = karatsuba(low1, low2)
    z1 = karatsuba((low1 + high1), (low2 + high2))
    z2 = karatsuba(high1, high2)

    return (z2 * 10 ^ m) + ((z1 - z2 - z0) * 10 ^ m2) + z0
```

5 Summary

Although the requirements of this project seem very simple, after careful consideration, however, one will find details such as handling uncertain number of command line arguments, entering non-integers and multiplying large numbers that require attention, and the fact that I used two versions of the code above to complete this project also shows that any qualified programmer must be careful about the document and iterative in identifying requirements when thinking about the problem.

In terms of the programming language, completing this project allowed me to deepen my understanding and mastery of what I learned in the course (e.g. the usage of iostream, argc and argv) and to explore the usage and features of several previously known but seldom used functions (such as atoi, memset and memcpy) to make the program performed as expected, and thus remember some of the things I needed to be aware of after using them incorrectly.