# C program report: Calculator

## Abstract:

Our project is a calculator, so this report is to state why we chose it as our project, how we analyzed and designed it, and how we implemented it. In addition, we list some problems we ever faced and how we solved it.

## Problem statement:

In our daily life, calculator is a very important tool for us students, but the calculators nowadays have some problems. Some of them are too simple to do calculation more than addition, subtraction, multiplication and division. Some of them are too complicated such as matlab that are not user-friendly and take plenty of time to run, so we are going to create one that not only fulfill the basic functions of calculators but also be easy to run.

## Group division:

**A**: his work is designing an interface, including creating a standard window, making sure that it can operate, creating buttons and boxes. At last he designed a way to integrate B and the C’s work.

**B**: he proposed our project, listed the work we should do and implemented the functions of statistics and linear equation. He also made powerpoints and writing lectures to explain our project.

**C**: he mainly implemented basic arithmetic operations, function operations and other small functions like finding false and conversion of number systems. At last, he also integrated the B’s work.

## Analysis:

To design a calculator, we wanted to achieve as more functions as we could, so before we started our project, we listed several core functions, as a guide to our project.

The first one is basic arithmetic operations. This is one of the most basic things in a calculator. Since the operations we write is a string and the computer scan the string, getting every element of it, so our problem is to tell the computer to know what is the meaning of every element in the operation. For example, while computer read a ‘(’ and a ‘)’, we should let it know it must calculate things between them first; While it read a ‘+’, it must know it means ‘+’, and it means it shall add the two numbers around the ‘+’. In addition, when operations exist ‘24’ or ‘sin’, the computer must connect them but remains ‘2’ and ‘4’ or ‘s’ ‘i’ ‘n’.

The second thing is functional operations. In fact, this thing is just like arithmetic operations. To calculate definite integral, we use its definition on math, which uses a minimum value: ‘0.001’, and then multiply it by the function’s value. All we should do is to get its value. So we store the function’s every element and change the ‘x’ in it to the number that you want to substitute to, and then it becomes basic arithmetic operations.

The third thing is about statistics. We decided to divide it into bivariate statistics and univariate statistics, including the average, variance, standard deviation. As for the univariate statistics, there are also regression equations, and uncertainties. So the best thing to calculate them is linked list.

The fourth thing is linear equations. We solve this problem by using its Cram Formula, which is an important theorem in math. As we all know, the value of a determinant is the sum of elements of a row or a column times their corresponding cofactors. And a cofactor is also a value of a determinant which is one order less than the original determinant. so we take advantage of the recursive algorithm.

Despite of these, we also design a interface. Having referred to some other calculators, we eventually chose the window instead of consol program. So we chose the API and the functions included to create a window. We set the window class structure, and then register class, in order to create windows according to the window class. The window maintains a message queue for each application. In our program, a "message loop" code can be used to get the message from the message queue. users check in his data, and click the button. the program will Get the Text from the InputBox, and stored as a string. then analysis & operate. The following are what we have designed:



What’s more, we add some other functions to rich our calculator. We added two other things in it. The first is conversion of numbers, which is to transform a decimal number to binary system, octonary system and hexademical system. The other thing is finding false, such as warning you that right brackets are not equal to left brackets.

## Design:

This is our whole flow chat:

The flow chat is from the viewpoint of users, as the flow chat says, we create a window, and users put their expressions into the box, the program substitute them into different functions according to different buttons that users press, and then the results are showed on another box. as for the specific functions we have used, we list them as following:

1. the interface

The window is a carrier of our project. Creating it has three major contains. First we initialized the window, including some essential functions of a window and evaluate several parameters to it, such as size, colors and something else. Second we made buttons and boxes (static text controls). Buttons are used to press and tell the program to operate certain functions; Boxes divided into two types: input boxes and output boxes. Then we used handles. The handle just likes a spokesman that tells what you are controlling and perform operations according to your signals. Third we simplified and optimized it, making it more user friendly, such as increasing some call-words to tell users how to use our calculator.

(2) the arithmetic operations

The essential part of arithmetic operations is to tell the compute the order of calculation. So we used a algorithm which is called Polish algorithm. This algorithm is used to change nifix expression into postfix expression. For example, ‘1\*(2+3)’ is a nifix expression and its postfix expression is ‘1 2 3 + \*’. The advantage of postfix is to convenience the computer to calculate. The tool to calculate it is stack. The specific process is as followings:

First you need four stacks. The first is Number stack; the second is Symbol stack; the third is Temporary stack; and the fourth is Calculator stack.

While inputting a expression as a string, it is scanned into several chars. These chars are push into the stacks in orders:

If it is a Number, then push it into Number stack;

If it is a ‘(’, then push it into Symbol stack;

If it is a ‘)’ then push all operators between the two brackets in Symbol stack into Number stack in orders.

If it is a common operator, and Symbol stack is empty, then push it into Symbol stack.

If it is a common operator, and the top element of Symbol stack is ‘(’, then push it into Number stack.

If it is a common operator, and the top element of Symbol stack is an operator as well and it has priority over the top operator, then push it into Symbol stack.

If it is a common operator, and the top element of Symbol stack is an operator as well and the top operator has priority over it, then push all elements in Symbol stack into Number stack in order until Symbol stack is empty or the top element is ‘(’ or the it has priority over the top operator.

If it is a common operator, and Symbol stack is empty, then push it into Symbol stack.

Having done all things above, push the rest elements in Symbol stack into Number stack(not including brackets). So at last the expression in Number stack is postfix expression.

The function of Temporary stack is to inverse the order of Number stack.

At last, fetch the top element stack of Temporary stack. If it is a number, then push it into Calculator stack; if it is a operator, then fetch top two numbers of Temporary stack and calculate these two numbers according to the operator.

There are one more thing we should do. When the expression has decimals or numbers that greater than 9, or operators like ‘sin’. Considering the traversal of the expression is some chars, and a char has only one element, so we designed a function to connect some of these chars, pushing some chars as a whole into Symbol stack or Number stack. So Symbol stack, Number stack, and Temporary stack are all string stacks. in this case, there comes another thing: we wrote a function to achieve the transition between the ‘float’ and ‘string’.

The following is the flow chat of this algorithm:

Start

Input expressions

Scan

expressions

Pop and push into Number stack

Push into symbol stack

Y

empty

operator

Y

N

N

Push into symbol stack

Y

Has priority over top operator

Push into Symbol stack

N

‘(’

N

Y

Top is ‘(’

Pop and push into Number

N

‘)’

End

N

Pop and push into Number

(3) Functional operation

We take advantage of stacks to do this operation, creating a stack to store every element of functional operation, and find the ‘x’ in it and then change it into the number you want to substitute into. At last, fetch the expression of the stack and substitute into arithmetic operation function.

The other thing is definite integral. We defined a minimum: 0.001, multiplied it by f(x), then add them from integral floor to integral upper.

The flow chats are as follows:

Input functions

N

Y

Start

Scan

expressions

Change to the number

Y

‘x’

N

N

Y

Function is over

Over

(4) statistics

Statistical computing is the main implementation of certain number of input and these Numbers are a series of processing, for univariate statistics, calculating the average, variance and the standard uncertainty of the bivariate statistics, in addition to calculating the average of each variable, the variance and the standard uncertainty, still need to the corresponding regression calculation and regression calculation of the uncertainty. Since we need to store any number of Numbers, we use a linked list structure to do this. A linked list is a structure of stored data and Pointers to the next cell. For the process of creating a linked list, it is necessary to first traverse the list to find the last cell (pointing to NULL), and then connect the new created linked list to achieve a variable length custom array. The process of statistical calculation is the process of traversing the linked list: starting from the first unit, p equals p minus >next, until the last one.

(5) For a system of linear equations of arbitrary elements: , the general solution can be expressed as



Where D is the determinant of the coefficient matrix, and  is the determinant of the i matrix whose column is replaced by a constant sequence. Therefore, the key to solving linear equations by using Cram's rule is to solve determinants. For a determinant, its value can be expressed as , where the cofactor  is a determinant of lower order than the original determinant. So you can compute the determinant using a recursive algorithm, and the initial first order determinant is going to be zero.

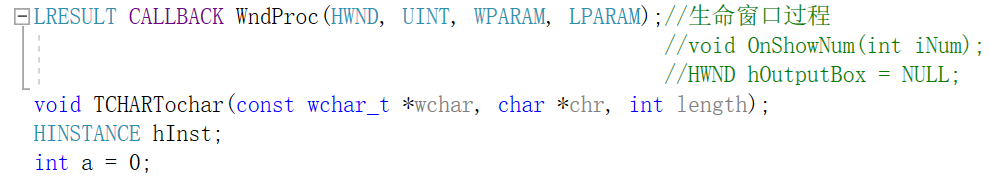
(6) other things

There are two extra things.

The first is conversion of number systems. It is an easy thing to do by using stacks. when you import a value, it is divided by 2 or 8 or 16 continuously, and push the remainder into the stack in order. At last fetch every element in the stack and connect them as the result.

The second is finding false. It is also an easy thing. We just wrote functions, which return false when the expression has something wrong, and popup a warning window.

## Implement:

1. interface:

Creating a window needs several steps which are showed as follows:

First we called some functions in API. In C language, seemingly simple operations can only be displayed after complex processing. For example, to open a file, first scan the hard disk, find the location of the file, and then read part of the data from the file, and put the data into the I/O buffer and memory. These data are 0 and 1 sequences, which are "translated" into characters against ASCII tables or Unicode tables, and then displayed on the monitor.

To simplify operations, Windows writes these complex operations in advance in a function called an API(Application Programming Interface).The header file <windows.h> declares most of the Windows that we need to use.

The second thing is about the main function in a Window. Our main program is different from the Console Application. Console programs take main() as the entry function, while Windows programs take WinMain() as the entry function. The WinMain function has four variables that define the current window handle, the previous window handle, the command-line arguments, and how the window should be displayed.

The third is about window structure. To create a window, you need to specify many properties of the window, many of which are generic and generally have the same value, so we set a structure: WNDCLASS instead and assign a value to it. The last field, lpszClassName, specifies the name of the current window class, which you pass to the CreateWindow() function to create a window based on that window class.

However, a window class is only a structure, and if we only define a structure variable, it will not be found by the value of the lpszClassName field when using. So you have to call RegisterClass() to register, to let the system know the name of the window class so it can be found the next time you use it.

The fourth is creating the window. With the window class, you can create Windows based on it. CreateWindows() function is used to create the window, with the name of the window class, window title, window style, window X-axis coordinate when initializing, window Y-axis coordinate when initializing, window width, window height and other parameters.

After the window is created through the CreateWindows() function, it simply allocates the memory space for the window and gets the handle, but the window does not display, so you also need to call the ShowWindow() function to display the window. The call to ShowWindow() only displays the window, but does not draw the client area, so we need to call the UpdateWindow() function, generate VM\_PAINT message, and draw the various controls in the client area.

The fifth is message loop. Windows maintains a message queue for each application, and when events occur, Windows automatically converts them into "messages" and posts them to the message queue. In our program, messages are retrieved from message queues through a "message loop" code. The GetMessage function is used to get a message from the message queue and save it to the MSG structure variable. Once the message is retrieved, the TranslateMessage function needs to be called to transform (translate) the message and then the DispatchMessage function is called to pass the message to the window procedure for processing (call the window procedure).

So the last thing is window procedure. Window procedures are functions that handle window events. Each time GetMessage gets a message, it is eventually thrown to the window process.

The window process has a parameter message, which will pass in the event type of occurrence. Different messages often need different processing, so the switch case statement is generally used to match.

The retrieved message can be processed or not, leaving Windows to process it by default .Return DefWindowProc() is a statement that tells Windows itself to handle messages that the application does not handle.

Window process can have window creation, window redraw, window destruction and other events. In the program, we used the CreateWindow function to create various function keys and data input and output window controls, and created a new logical font with CreateFont to beautify the interface.

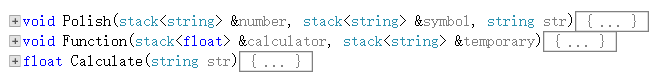
These window controls are static, which can display text; Some are edit types, which can be used for inputting; There are buttons to click on to trigger events.

When the button is clicked, an event is triggered to generate a WM\_COMMAND message, which is delivered to a message queue and then processed. Different buttons correspond to different processing methods, but basically they all involve taking input information - converting data type - processing data - converting data type - output information. And there are also type conversion that is because our program displays Unicode, which is a Wide Character that cannot be directly mixed with char and string, so format conversion is required for input and output. The converted characters are stored as strings and are computed and output, and the format conversion will also be stated below.



(2) Basic arithmetic operation:

As we have designed, we uses Polish algorithm and four stacks to implement our program. The major things are three functions:



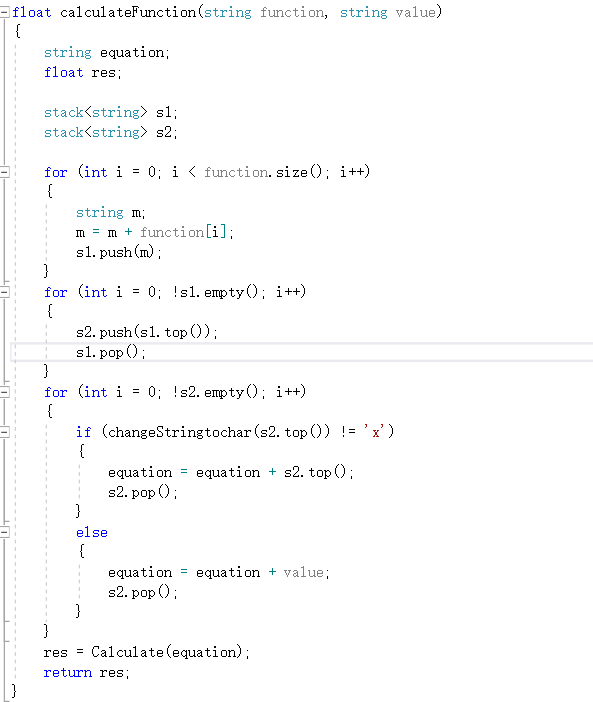
The Polish function is to get a postfix expression, and it is stored in the Number stack. The Calculate function is to calculate as it is called, the final result is in Calculator stack(the top of the stack). The Function function is to tell the computer how to calculate numbers in the Calculator stack.

There are some other functions, which reflect some major problems we have faced during our implementation. The two major things we have faced are how to connect numbers and symbols and how to achieve the transition between float and string.

This is connecting function, it is very common. We just connect several chars to a string. Like connect ‘2’ and ‘4’ to ‘24’. And this function can operate on decimals.

(not the whole function)

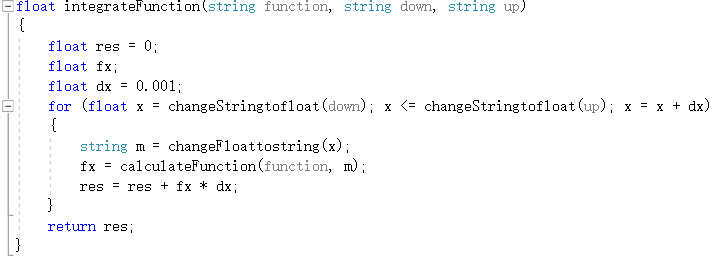
The next are transition functions. This time we use a ingenious method. We transform float to string by getting every number in each bit and change it into a char and then connect these chars to a string. The method we change string to float is just like the inverse process. These two functions are very useful. In the end we need transform the result into a string to output, and almost all of our functions’ result are float, so we can easily change our result to string. Besides, the string type is convenient to use but when need calculating, we must change it to a float.



(3) Functional operation

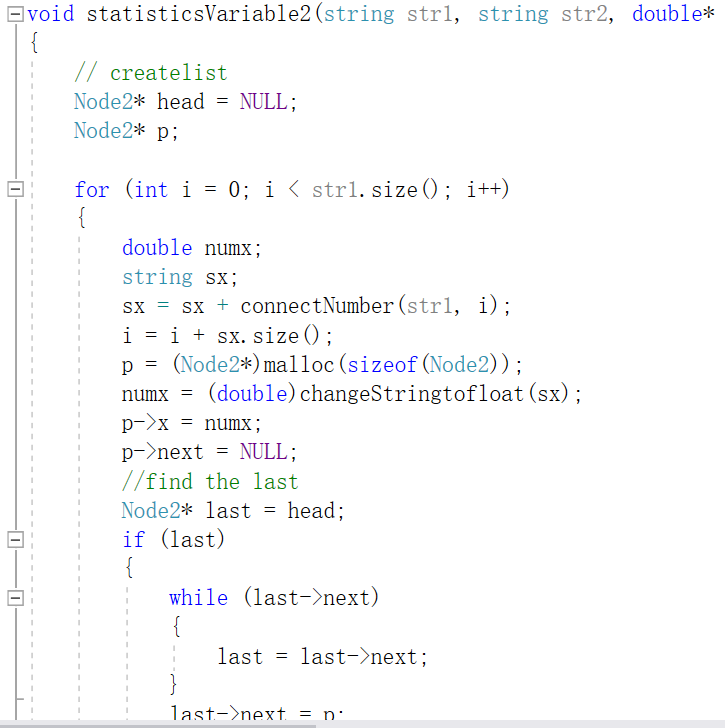
In this program, we use another stack to store the function, and while you import a value of ‘x’, first it change ‘x’ to string and then scan the function and take place of ‘x’ and push the operation into a stack again, then substitute the expression into Calculate function.

As for definite integral, we just define a ‘dx’ : 0.001, and take advantage of its definition in math.



(4) statistics

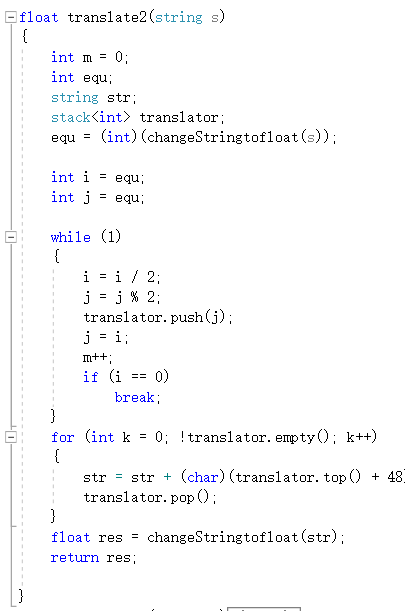
As we have designed, we uses linked list to store variables. The difficulty encountered is in the regression calculation. For example, if there are only two regression variables, the value of a certain uncertainty will theoretically tend to be positive infinity, so it is necessary to make an out-of-bounds judgment in the calculation.

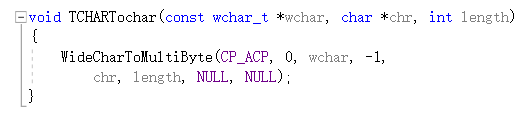


(5) linear equations

Having discussed the method we will use, implementing this function is an easy thing. What we should do is use the recursive algorithm correctly. Considering the order of coefficient matrix is not certain so we need store the information in a binary array of a given size. This requires binary array dynamic memory allocation. Unlike unary arrays, binary arrays need to allocate rows first, using malloc, and then allocate the columns of each row iteratively. The final free process requires free columns for each row, then free rows.

(6) This is one of transition functions. We define two integer variable: i and j. after each division, i is used to store the integer part of the quotient, and j is used to store the remainder. A stack is used to store these remainders.

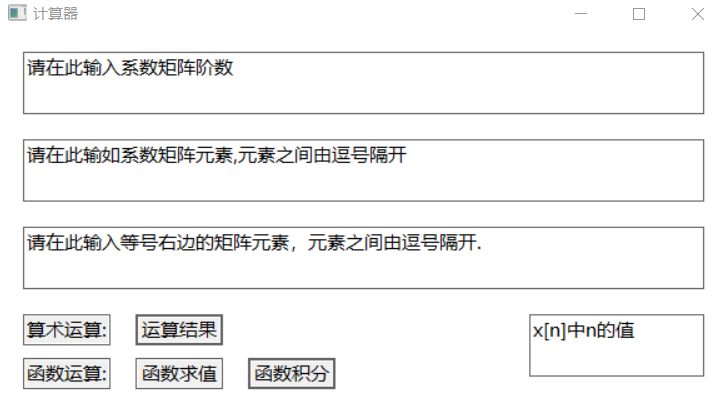


(7) Having done all things above, we at last integrated them to one single project. The main problem we ever faced was that Ren Xiaoyu and Guan Hongyi used console program but Ma yu used windows. So how could we fetch expressions of boxes to substitute into our functions. In fact, when we exchanged our codes, there were always some false. To solve this problem, we ultimately changed all functions that they all use strings, and the following is the function that fetch the expression of boxes and change it into a string.

In fact, during the process of integration, the basic arithmetic operations and function operations is relatively easy, because this two functions use string all the time, and it is convenient to combine it to the window program. A major problem we have faced is Guan Hongyi’s program are full of “ptintf” and “scanf” but these two statements are not compatible with the window program, so we need delete all these statements and write new statements to make it understood by the window program.

First we check the codes about statistics. Considering it uses linked list so we must make it sure the linked list is not destroyed. So we store the varieties in a string and each variety is divided by a dot, then we scan the string and fetch each variety expect for dots by using our connecting number function. Next we push each variety into the linked list in order.

Then we check the codes about linear equations. In fact in this part we can’t do it perfectly because in the end, the result which is every variable(x[1],x[2]...) is output once by once and our window program’s boxes can’t do this. It only can output one result or output all results once a time, but there comes another problem, which is the coefficient matrix’s order is depend on users, so we can’t output a series of results with uncertain numbers. So we design another input box to control which variable you want to get. So the eventual version is :

As the picture shows, when we solve linear equations, we need four input boxes. The first is to store the order of coefficient matrix. The second one is about the specific element of coefficient matrix, and in this box, we can’t input elements as usual, we just can input all elements in a line, and must make it the right order. Taking the two dimensional system equation for example, the order you input is a[11],a[12],a[21],a[22]. In this case, the third box is b[1],b[2], which are elements in the right of equal signs. In the last box, if you input ‘1’, you will get the value of x[1]. Seemly, if you input ‘2’, you will get the value of x[2].

After integrating all of our work, considering the beauty of the interface, we revise the coordinates of boxes and buttons, and wrote some call-words to convenience users and be more user-friendly. The final version is as follows:



In our program, when you press left buttons, it will appear some call-words in input boxes, telling you how to do next. When you press right buttons, our program will analyze and operate what you have input and give you the result in the output box bellow.



## Text:

Having done all the things we should do, then we tested our project. Sometimes several bugs appeared, but generally it is good enough, and the result is normal. Here are some screenshots of our outcomes.

Basic arithmetic operations Definite integral





Bivariate statistics linear equations

## Conclusion:

This is the first time for us to design and implement a computer program by ourselves. It is a challenge, especially for us students who have rarely foundation in computer programming. But at last, we achieve our goals successfully through our effort, perseverance, some cleverness and a little luck.

During this period, our eventual goal is clear, and our respective goal is also clear. Under the condition of this, having seen other calculators, A chose to take the advantage of windows as our carrier; B select inked list, a flexible structure, to operate statistics and designated Cram formula and recursive algorithm; C chose the Polish algorithm, a powerful algorithm, and stacks a powerful structure, to calculate basic arithmetic operations and use it to achieve function operations, as well as conversion of number systems. At last, we used strings to be a connection between the interface and specific functions.

Certainly, we faced lots of problems, either. C was confused by the transition between floats and strings, and eventually created a function to implement this. When integrating, we found that our program had errors all the time, but we solved them one by one.

During this programming process, we command how to operate some structures and how to integrate our work. More importantly, we have learned what is a project, what should we do before we start specific steps, how we divide the whole program into several small things to every member. We all think these are what we need to learn indeed.

The C program tells us a lot.