绘图板实验报告

一、 编译运行方法

在本压缩包的 dev 和 ACLlib/sample/cpp 中有可用 dev-cpp 和 VS 运行的两个版本。运行后，可在命令行窗口按照样例输入函数，画出一次图像后函数会自动结束。

二、 程序功能

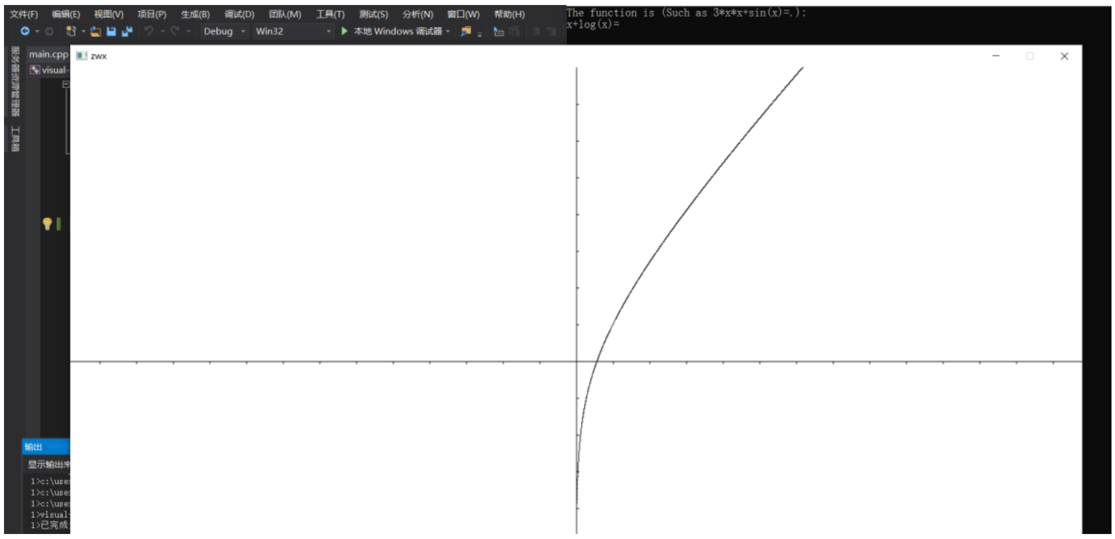
在命令行窗口输入一个带有 x 的表达式，本程序可在图形窗口绘制出函数图像。例如，输入 sin(x)+x*x=。中间可以有空格。注意，不能省略等号及使用中文输入法。横轴、纵轴的端点间代表的长度为 1。

支持的绘图函数：

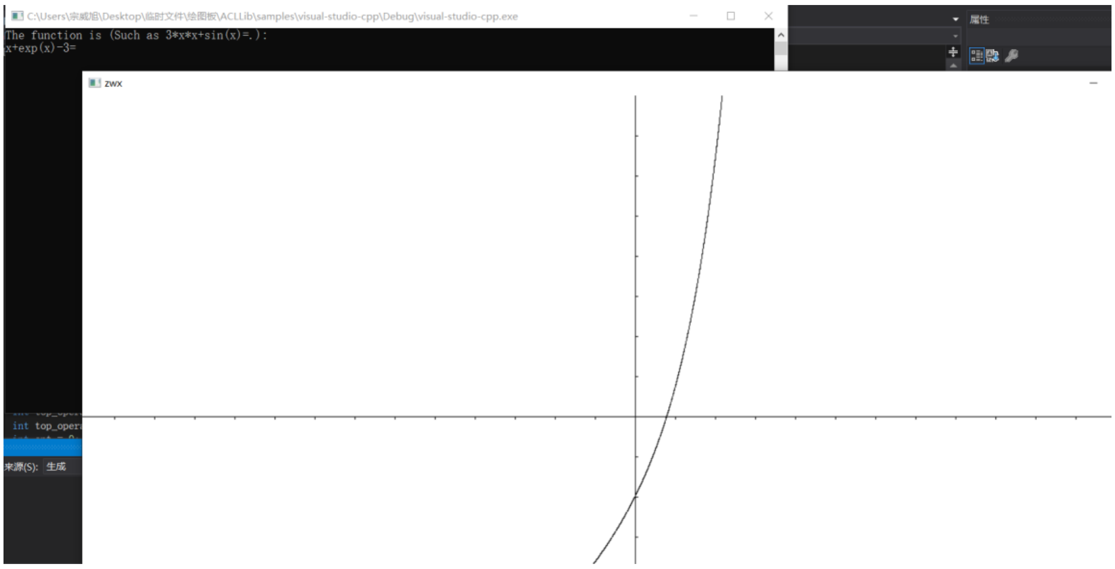
初等函数，比如幂函数、指数函数（exp(expression)），对数函数（exp(expression)），三角函数（sin(expression)）、反三角函数（asin(expression)）、绝对值函数（fabs(expression)）。

程序存在一些不能支持的功能。比如程序不支持自动调节比例，且只能输入一定区间范围和一定值域范围内的值。不支持隐函数（题目要求好像也没有办法输入隐函数）、积分、导数等功能。

运行中的样例如下：



```
The function is (Such as 3*x*x+sin(x)=.):
x+log(x)=
```


C:\Users\宗威旭\Desktop\临时文件\绘图板\ACLLib\samples\visual-studio-cpp\Debug\visual-studio-cpp.exe
zwx
属性
int top_oper
来源(S): 生成

```
The function is (Such as 3*x*x+sin(x)=.):
x+exp(x)-3=
```

三、　心得体会

本次作业实际上很大程度上可以利用第一次大作业的计算器。因为不要求绘制隐函数，所以没有必要扫描所有的点，只需计算对应值即可。但是由于第一次做的计算器可移植性与可扩展性都很差，所以在这次作业对上次的计算器做了很大改动。包括采用函数指针优化以及函数结构优化。

本次画图中由于不了解 ACLLib 的内部实现原理，在完成代码后，图形窗口总是显示不出函数值。后来经过调试，能以一个固定比例画出函数图像。我曾经试过根据最值计算比例来绘制图像，但由于……某些过于巨大的函数还不能够很好的处理，会导致 ACLLib 运行时崩溃，且也没有足够的时间来设置相应区间以减缓扫描点的压力。在这些细微的功能上还有很大的改进的空间。所以，等考试周会在 GitHub 上推上改进版本。

第二次大作业收获还是比较大的，毕竟计算机小白表示这些都没学过，但还是简单学一学可以实现的。觉得还有很大很大的提升空间。只是，还是要系统的学一学堆栈、队列这些算法，直接应用对我来说还是不能够理解这个算法还能做些什么。二叉树结构好像可以更好的保存这次的表达式，但是过于菜

鸡，没能实现。数据结构也还要加强。

四、 源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "acllib.h"

#define length 1380
#define width 800
#define Ox length/2
#define Oy width/2
#define delta 0.1

#define SIZE 1000
#define MAX 20
#define SIN 1
#define COS 2
#define SQRT 3
#define FABS 4
#define TAN 5
#define ASIN 6
#define ACOS 7
#define ATAN 8
#define EXP 9
#define LOG 10

int top_operand = -1;
int top_operator = -1;
int cnt = 0;
int judge = 1;
int rate = 10;
double number = 0;
int flag = 0, dot = 1;
int funcnt = 0;
int function[MAX] = { 0 };
char InfixExpression[SIZE];
char *p, *t;

void DrawXYCoordinate();
double ReadFunction(double x);
```

```
void PrintIntroduction();
void Error();
int TransToDigit();
void InsertOperand(double Operand[]);
void InsertOperator(char Operator[]);
void DealOperand(double Operand[], char Operator[]);
int CompareOperator(char Operator[]);
void DealOperator(double Operand[], char Operator[], int i);
void Calculation(double Operand[], char Operator[]);
void DealFunction(double Operand[], char Operator[]);
double PrintResult(double Operand[], char Operator[]);

double(*funcp[15])(double x) = { 0,sin,cos,sqrt,fabs,tan,asin,acos,atan,exp,log };

int Setup()
{
	int max, min;
	initConsole();
	PrintIntroduction();
	gets(InfixExpression);
	double x,y,dx,dy;
	max = 1;
	for (dx = 0;dx < length;dx += delta)
	{
		x = (dx - Ox) / rate;
		y = ReadFunction(x);
		if (fabs(y) > max) max = y;
		dy = Oy - y * rate;
	}
	rate = width / (10 * max);
	DrawXYCoordinate();
	beginPaint();
	for (dx = 0;dx < length;dx += delta)
	{
		x = (dx - Ox) / rate;
		y = ReadFunction(x);
		dy = Oy - y * rate;
		putPixel(dx, dy, DEFAULT);
	}
	endPaint();
	return 0;
}

void DrawXYCoordinate()
```

```
{
	initWindow("Draw", DEFAULT, DEFAULT, length, width);
	const int point = 4;
	int x, y;
	beginPaint();
	line(0, width / 2, length, width / 2);
	line(length / 2, 0, length / 2, width);
	for (x = Ox, y = width / 2;x < length;x += rate)
	{
		moveTo(x, y);
		lineRel(0, point);
	}
	for (x = Ox, y = width / 2;x > 0;x -= rate)
	{
		moveTo(x, y);
		lineRel(0, point);
	}
	for (x = length / 2, y = Oy;y < width;y += rate)
	{
		moveTo(x, y);
		lineRel(point, 0);
	}
	for (x = length / 2, y = Oy;y > 0;y -= rate)
	{
		moveTo(x, y);
		lineRel(point, 0);
	}
	endPaint();
}
double ReadFunction(double x)
{
	double Operand[SIZE] = { 0 };
	char Operator[SIZE] = { 0 };
	double result = 0;

	top_operand = -1;
	top_operator = -1;
	flag = 0;dot = 1;
	p = InfixExpression;
	if (*p == '*' || *p == '/') Error();
	else if (*p == '+' || *p == '-')
	{
		InsertOperator(Operator);
		top_operand++;
```

```
        p++;
    }
    for (t = p + 1;*p != '=';p++, t++)
    {
        if (*p == ' ') continue;
        if (isdigit(*p)) DealOperand(Operand, Operator);
        else if (*p == '(' && (*t == '+' || *t == '-'))
        {
            InsertOperator(Operator);
            p++;t++;
            InsertOperator(Operator);
            top_operand++;
        }
        else if (((*p == '+' || *p == '-' || *p == '*' || *p == '/') && (*t != '+'&&*t != '-'&&*t !=
'*'&&*t != '/')) || (*p == '('&&*t != ')') || *p == ')')
        {
            int i = CompareOperator(Operator);
            DealOperator(Operand, Operator, i);
        }
        else if (*p == 'x')
        {
            number = x;
            InsertOperand(Operand);
        }
        else if (isalpha(*p)) DealFunction(Operand, Operator);
        else Error();
    }
    result = PrintResult(Operand, Operator);
    return result;
}
void PrintIntroduction()
{
    if (cnt) printf("The function is:\n");
    else printf("The function is (Such as 3*x*x+sin(x)=.):\n");
}
void Error()
{
    printf("Syntax ERROR!\n");
    top_operand = -1;
    top_operator = -1;
    exit(0);
}
int TransToDigit()
{
	int n;
	n = *p - '0';
	return n;
}
void InsertOperand(double Operand[])
{
	number = number / dot;
	Operand[++top_operand] = number;
	number = 0;flag = 0;dot = 1;
}
void InsertOperator(char Operator[])
{
	Operator[++top_operator] = *p;
}
void DealOperand(double Operand[], char Operator[])
{
	if (!isdigit(*t) && *t != '.')
	{
		number = number * 10 + TransToDigit();
		if (flag) dot *= 10;
		InsertOperand(Operand);
	}
	else if (*t == '.')
	{
		number = number * 10 + TransToDigit();
		flag = 1;
		p++;t++;
	}
	else
	{
		number = number * 10 + TransToDigit();
		if (flag) dot *= 10;
	}
}
int CompareOperator(char Operator[])
{
	int i;
	if (*p == ')') i = 1;
	else if (top_operator == -1 || (Operator[top_operator] == '+' || Operator[top_operator]
== '-') && (*p == '*' || *p == '/') || Operator[top_operator] == '(' || *p == '(') i = 0;
	else i = 2;
	return i;
}
void DealOperator(double Operand[], char Operator[], int i)
```

```
{
    if (i == 0)     InsertOperator(Operator);
    else if (i == 1)
    {
        while (Operator[top_operator] != '(')
        {
            Calculation(Operand, Operator);
        }
        top_operator--;
        if (!function[funcnt])
        {
            Operand[top_operand] = (*funcp[function[funcnt -
1]])(Operand[top_operand]);
            function[--funcnt] = 0;
        }
    }
    else
    {
        Calculation(Operand, Operator);
        while (CompareOperator(Operator) == 2)
        {
            Calculation(Operand, Operator);
        }
        InsertOperator(Operator);
    }
}
void Calculation(double Operand[], char Operator[])
{
    double m = 0;
    if (Operator[top_operator] == '/')
    {
        if (Operand[top_operand] == 0) Error();
        m = Operand[top_operand - 1] / Operand[top_operand];
        top_operator--;top_operand--;
        Operand[top_operand] = m;
        return;
    }
    else if (Operator[top_operator] == '+') m = Operand[top_operand - 1] +
Operand[top_operand];
    else if (Operator[top_operator] == '-') m = Operand[top_operand - 1] -
Operand[top_operand];
    else if (Operator[top_operator] == '*') m = Operand[top_operand - 1] *
Operand[top_operand];
    top_operator--;top_operand--;
    Operand[top_operand] = m;
}
void DealFunction(double Operand[], char Operator[])
{
    char temp[10] = { 0 };
    int i = 0;
    while (isalpha(*p))
    {
        temp[i++] = *p;
        p++;t++;
    }
    if (!strcmp(temp, "sin")) function[funcnt++] = SIN;
    else if (!strcmp(temp, "cos"))function[funcnt++] = COS;
    else if (!strcmp(temp, "sqrt"))function[funcnt++] = SQRT;
    else if (!strcmp(temp, "fabs"))function[funcnt++] = FABS;
    else if (!strcmp(temp, "tan"))function[funcnt++] = TAN;
    else if (!strcmp(temp, "atan"))function[funcnt++] = ATAN;
    else if (!strcmp(temp, "asin"))function[funcnt++] = ASIN;
    else if (!strcmp(temp, "acos"))function[funcnt++] = ACOS;
    else if (!strcmp(temp, "exp"))function[funcnt++] = EXP;
    else if (!strcmp(temp, "log"))function[funcnt++] = LOG;
    else Error();
    p--;t--;
}
double PrintResult(double Operand[], char Operator[])
{
    if (top_operand != -1)
    {
        while (funcnt)
        {
            Operand[top_operand]          =          (*funcp[function[funcnt          -
1]])(Operand[top_operand]);
            function[--funcnt] = 0;
        }
        while (top_operand != 0)
        {
            Calculation(Operand, Operator);
        }
        return Operand[0];
    }
    else Error();
}
```