浙江大学实验报告

专业：混合班
姓名：徐圣泽
学号：3190102721
日期：2020.4.13

课程名称：C 程序设计专题　　指导老师：翁恺　　实验名称：函数绘图

一、实验要求

（1）实现一个GUI程序，从标准输入读入一个`f(x)`的函数表示，如`2x^2-6x+20`，其中幂次用`^`表示，系数和`x`之间不需要乘号，最高幂次不限。

（2）读入x的区间的起点和终点，如`-1.4 1.4`。

（3）根据函数表达式和定义域计算值域并确定`y`轴的范围，在窗口中画出函数图象和坐标轴。

（4）坐标轴要求画出原点和两条轴线，轴线上标出合理的坐标点。

二、实验思路与过程描述

实验思路分为四个部分：读入函数表达式、计算函数值、确定坐标轴、画函数图像。

一、读入函数表达式：逐项读入字符，链表中存储多项式各项的系数和指数。

二、计算函数值：遍历，对链表每一项计算值并最终求和累加。

三、确定坐标轴：通过函数值域和定义域的区间，确定坐标轴的范围，再补充网格线。

四、画函数图像：利用`epsilon`（极小量，精度足够小）进行遍历，画出每个点，“描点”法绘制函数图像，文本形式输出函数的最大值和最小值。

三、实验代码解释

3.1 `main.c`

`main`函数需要主要是为了完成实验的第三个部分，即通过函数定义域和值域求得坐标轴的范围。

本函数最终绘制的图象中坐标轴的位置是固定的——原点位于界面的中心，X轴和Y轴分别位于高与宽的中间，故原点两侧的坐标是一样的。因此在本程序中，只需要在坐标轴合适的位置标注坐标即可，因此本部分需要求出坐标轴坐标所需要的范围。

```
    double X=fabs(left);
    if(fabs(right)>fabs(left)){
        X=fabs(right);
    }
    X=X*1.5;
```

X轴的坐标范围计算，比较`left`和`right`的绝对值，取较大的一个扩大`1.5`倍，这样使得函数图象可以保持在界面较中间的部分。

```
    double i,max=funcvalue(f,left),min=funcvalue(f,left);
    double Y=fabs(min);
    double m=left,n=left;
    for(i=left;i<=right;i+=epsilon){
```

```
        if(funcvalue(f,i)>max){
            max=funcvalue(f,i);
            m=i;
        }
        if(funcvalue(f,i)<min){
            min=funcvalue(f,i);
            n=i;
        }
    }
    if(fabs(max)>fabs(min)){
        Y=fabs(max);
    }
    Y=Y*1.5;
```

Y轴的坐标范围计算，原理与前面关于X轴的计算类似，调用了求值函数（在后面会解释该部分）遍历求值并比较得到了Y轴所需要的坐标范围。上面的代码其实可以进行一定程度的简化，直接遍历逐项求值并得到绝对值的最大值。

我在这个部分求出原函数的最大值和最小值，并保留了在`x`取何值时函数得到最大值和最小值，将数据存于`m`和`n`中，并在后面将这两个值传入`void paintvalue(func*f,double m,double n,double X,double Y)`函数中，以便于在函数图象界面以文本的形式输出其最大值和最小值。

```
    beginPaint();
    funcimage(f,left,right,X,Y);//绘制函数图象
    paintaxis(X,Y); //绘制坐标轴
    paintvalue(f,m,n,X,Y);//标注函数的最大值和最小值
    endPaint();
```

除此之外，还需要对程序中所用到的一些参数进行宏定义：

```
#define width 800
#define height 800
#define epsilon 0.000001
```

`main`函数剩下的部分便是绘图部分，这几个函数在下面部分着重介绍。

3.2 `function.h`

```
#ifndef _FUNCTION_H_
#define _FUNCTION_H_

typedef struct _func func;

func* creatfunc();//f(x)表达式读入函数
double funcvalue(func *f,double x);//求值函数
void funcimage(func *f,double left,double right,double X,double Y);//图像绘制函数
void paintvalue(func*f,double m,double n,double X,double Y);//输出最值函数
void paintaxis(double X,double Y);//坐标轴绘制函数

#endif
```

这个部分是比较熟悉的函数声明和类型声明以及编译预处理。

3.3 `function.c`

这个部分主要分为链表的前向声明和几个函数的定义：①表达式的读入函数`func* creatfunc();`、②求值函数`double funcvalue(func *f,double x);`、③绘制图象函数`void funcimage(func *f,double left,double right,double X,double Y);`、④在界面中输出最值的函数`void paintvalue(func*f,double m,double n,double X,double Y)`、⑤坐标轴绘制函数`void paintaxis(double X,double Y);`。

（一）链表

```
typedef struct _func{
    double coef;
    double exp;
    struct _func *next;
} func;
```

定义了`double`类型的`coef`和`exp`，用来存储函数各项的系数和次数。

（二）表达式的读入函数

```
    if(op=='+'){
        p->coef=1.0;
    }
    else{
        p->coef=-1.0;
    }
```

这个是`while`循环的第一个部分，`while`循环的判断条件是`op`是否为`\n`，因此在循环进入这个部分之前需要提前定义`char op='+';`且定义`p->coef=1;`在循环的过程中，如果`op`是`'-'`，则`p->coef`赋值为`-1`。

```
    flag=scanf("%lf",&value);
    if(flag==1) p->coef*=value;
```

这个部分需要判断输入的是否是一个浮点数，利用`scanf`的返回值，如果返回值为`1`，则将`p->coef`乘上输入的浮点数则得到了多项式这一项的系数，正负的判断已经在此前进行。

这次实验也是我第一次学习到`scanf`返回值的用法：`scanf`函数返回成功读入的数据项数，读入数据时遇到了“文件结束”则返回`EOF`。举例说明，`scanf("%d %d",&a,&b);`当`a`和`b`都被成功读入时，`scanf`的返回值是2；如果只有`a`被成功读入，返回值为1；如果`a`和`b`都未被成功读入，返回值为0；

```
    if((ch=getchar())=='x'){
        if((ch=getchar())=='^'){
            flag=scanf("%lf",&value);
            if(flag==1) p->exp*=value;
            ch=getchar();
        }
    }
    else{
        p->exp=0.0;
    }
    op=ch;
```

这个部分为了判断输入的字符，并根据不同情况作出不同的反应。首先判断字符是否是`x`，如果不是，则该项为常数项；如果是，则继续判断是否为`^`，读入该项的次数并与`p->exp`相乘，注意`p->exp`的初始值应在此之前赋值为`1`。

```
    if(head){
        n->next=p;
        n=n->next;
    }
    else{
        head=p;
        n=p;
    }
```

函数最后部分是常规的操作，链表指向下一项继续循环，这里的`n`是在循环之前定义的`func *n=head;`而`p`是循环内部定义的`func *p=(func*)malloc(sizeof(func));`这里的原理与之前的`Linked List`相同。

注：这个函数部分需要实名感谢赵泓珏同学的代码，我在写函数的过程中受到了他的启发，但将双向链表改成了单向链表，比较简洁，对我而言也比较熟悉。在此之前我在读入函数部分卡了很久还是没有写出来，导致后半部分的进度停滞。后来从他的推荐作业中学习到了读入函数的方法，因此这部分的代码原理和他是较为相似的。但在调试的过程中，也发现了这个部分的一些不足之处，这在最后的实验体会部分细说。

（三）求值函数

```
double funcvalue(func *f,double x){
    func* p;
    double sum=0.0;
    for(p=f;p;p=p->next){
        sum+=p->coef*pow(x,p->exp);
    }
    return sum;
}
```

这个函数的思想与`P1`中的`Linked List`中的很多函数思想是一致的——遍历。`p->coef*pow(x,p->exp)`就是每一项的值，依次累加求和即得到了函数值。注意`double sum=0.0`，这是每一次需要用到求和时都要提醒自己的事情。

（四）函数图像的绘制

```
void funcimage(func *f,double left,double right,double X,double Y){
    double x,y,x1,y1;
    double k1,b1,k2,b2;
    for(x=left;x<=right;x+=epsilon){
        y=funcvalue(f,x);
        k1=width/(2*X);
        b1=width/2;
        k2=-height/(2*Y);
        b2=height/2;
        x1=k1*x+b1;
        y1=k2*y+b2;
        putPixel(x1,y1,RED);
    }
}
```

这个函数最关键的部分就是关于点的坐标变换问题——两个坐标系原点的位置和正方向均不同，因此需要变换坐标。我们所输入的自变量`x`的值和计算得到的`y`的值都是相对于我们的目的坐标系，而非系统原本的坐标，但这两个坐标存在线性的关系，因此可以通过待定系数的方法解得两个坐标的线性关系并解得对应的系统坐标系下的坐标。

$$\begin{cases} width = k_1 X + b_1 \\ 0 = -k_1 X + b_1 \end{cases}$$

$$\begin{cases} 0 = k_2 Y + b_2 \\ height = -k_2 Y + b_2 \end{cases}$$

上面就是坐标代换的方程组，这里的`width`和`height`都已宏定义为`800`，分别为X和Y的坐标变换，解得待定系数并在代码中进行坐标变换。

在得到了坐标后，通过`putPixel(x1,y1,RED);`描出各个点，因为`epsilon`足够小，因此绘制的点可以大致构成一个函数图像，且精度较高。

```
void paintvalue(func*f,double m,double n,double X,double Y){
    double x1,x2,y1,y2;
    char str1[100];
    char str2[100];
    x1=width/(2*X)*m+width/2;
    y1=-height/(2*Y)*funcvalue(f,m)+height/2;
    x2=width/(2*X)*n+width/2;
    y2=-height/(2*Y)*funcvalue(f,n)+height/2;
    sprintf(str1,"%s%.2f%s%.2f","当x=",m,"时 fmax=",funcvalue(f,m));
    sprintf(str2,"%s%.2f%s%.2f","当x=",n,"时 fmin=",funcvalue(f,n));
    setTextSize(10);
    setTextColor(RGB(172,176,234));
    paintText(x1,y1,str1);
    paintText(x2,y2,str2);
}
```

因为在进行关于函数值的计算时，常常需要得到它的最值，因此我想，不妨在绘制函数图像时一并将最值显示出来。因此这个部分的代码的功能便是为了在最终的函数图像旁输出函数的最大值和最小值，分别位于取到最大值和最小值的`x`旁。与上面同理，这个部分的`x`和`y`坐标也需要进行坐标的变换，原理也与上面一样。

这个部分的函数过程中，用到了一个新的函数，即`sprintf`函数。

函数声明：`int sprintf(char *str, const char *format, ...);`

参数部分：

`str`这是指向一个字符数组的指针，该数组存储了字符串。

`format`这是字符串，包含了要被写入到字符串`str`的文本。它可以包含嵌入的`format`标签，`format`标签可被随后的附加参数中指定的值替换，并按需求进行格式化。

附加参数：根据不同的`format`字符串，函数可能需要一系列的附加参数，每个参数包含了一个要被插入的值，替换了`format`参数中指定的每个`%`标签。参数的个数应与`%`标签的个数相同。

返回值：如果成功，则返回写入的字符总数，不包括字符串追加在字符串末尾的空字符。如果失败，则返回一个负数。

因此在这个函数中定义了字符串数组`char str[]`，将其余数据类型转化成字符串。

原本我以为这个函数只能格式化一种数据类型，但后来发现，只要在`format`中的参数的数据类型和附加参数中的数据类型一一对应即可，附加参数中的各个参数用`,`隔开。这个函数在下面的代码中还会出现，不过使用的情况比这里要简单。

（五）坐标轴绘制

```
    setPenColor(RGB(0,0,0));
    setPenWidth(2);
    line(0,height/2,width,height/2);//x轴线段
    line(width,height/2,width-10,height/2-5);//箭头
    line(width,height/2,width-10,height/2+5);//箭头
    line(width/2,0,width/2,height); //y轴线段
    line(width/2,0,width/2-5,10);//箭头
    line(width/2,0,width/2+5,10);//箭头
    paintText(width/2+5,height/2+2,"O");
    paintText(width-15,height/2+6,"x");
    paintText(width/2+10,0,"y");
    setTextSize(20);
    paintText(10,10,"函数图像");
```

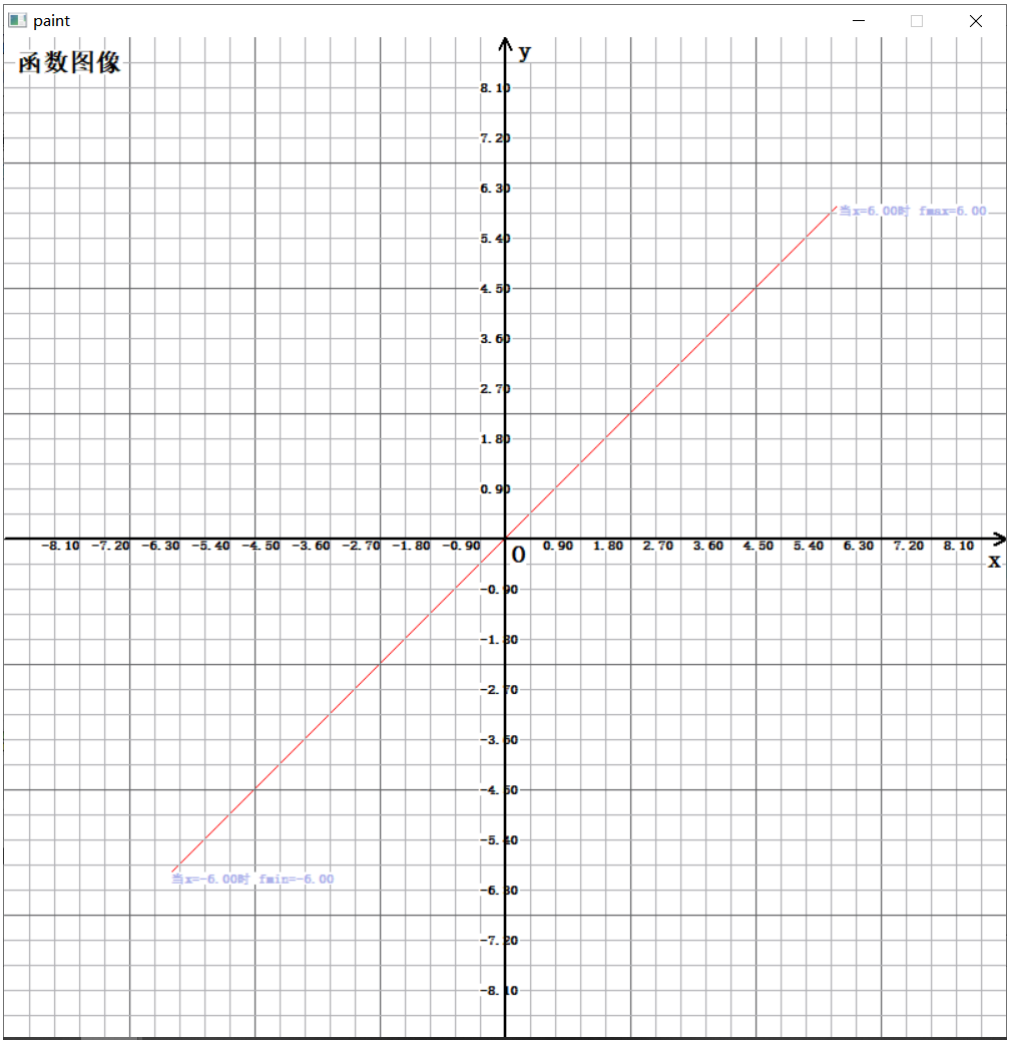
坐标轴绘制函数最基础的部分是画出两条“带箭头”的线段作为坐标轴。这里的`width`和`height`都已经宏定义。同时这个函数在界面合适的位置显示原点和坐标轴`x`和`y`的标注。

```
    int i,w=width/40;
    char str1[100];
    for(i=w;i<width;i+=w){
        if(i%(width/8)!=0&&i!=width/2){
            setPenColor(RGB(179,179,183));
            line(i,0,i,height);
        }//绘制浅灰色网格线
        if(i%(width/8)==0&&i!=width/2){
            setPenColor(RGB(102,103,105));
            line(i,0,i,height);
        }//绘制深灰色网格线
        sprintf(str1,"%.2f",(2*X/width)*i-X);
        if(i%(width/20)==0&&i!=width/2){
            setTextSize(10);
            paintText(i-10,height/2,str1);
        }//标注坐标轴刻度
    }
```

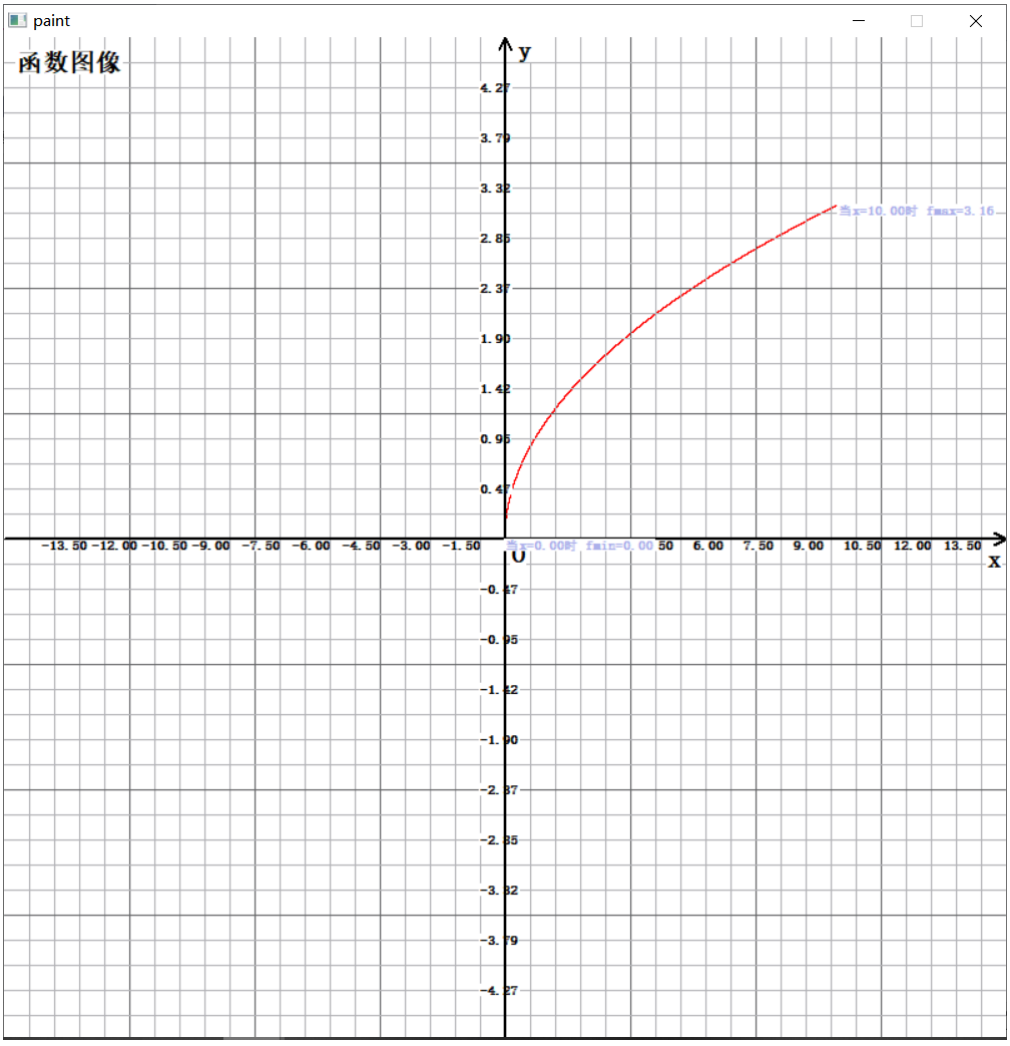
接下来的一部分是除了坐标轴外的网格线的绘制，上面展示的代码是纵向（`y`轴方向）的网格线绘制过程。考虑到美观程度，界面中每五条线都会出现一条深灰色线，其余的网格线是浅灰色线。除此之外，需要隔适当的距离在坐标轴上标注出刻度，我设置的是每隔两格显示一次刻度。这里同样用到了上面所提到的`sprintf`函数，但情况较为简单，只需要转换一个`double`的数据类型。`x`轴防线的网格线绘制与此类似。

3.4样例展示

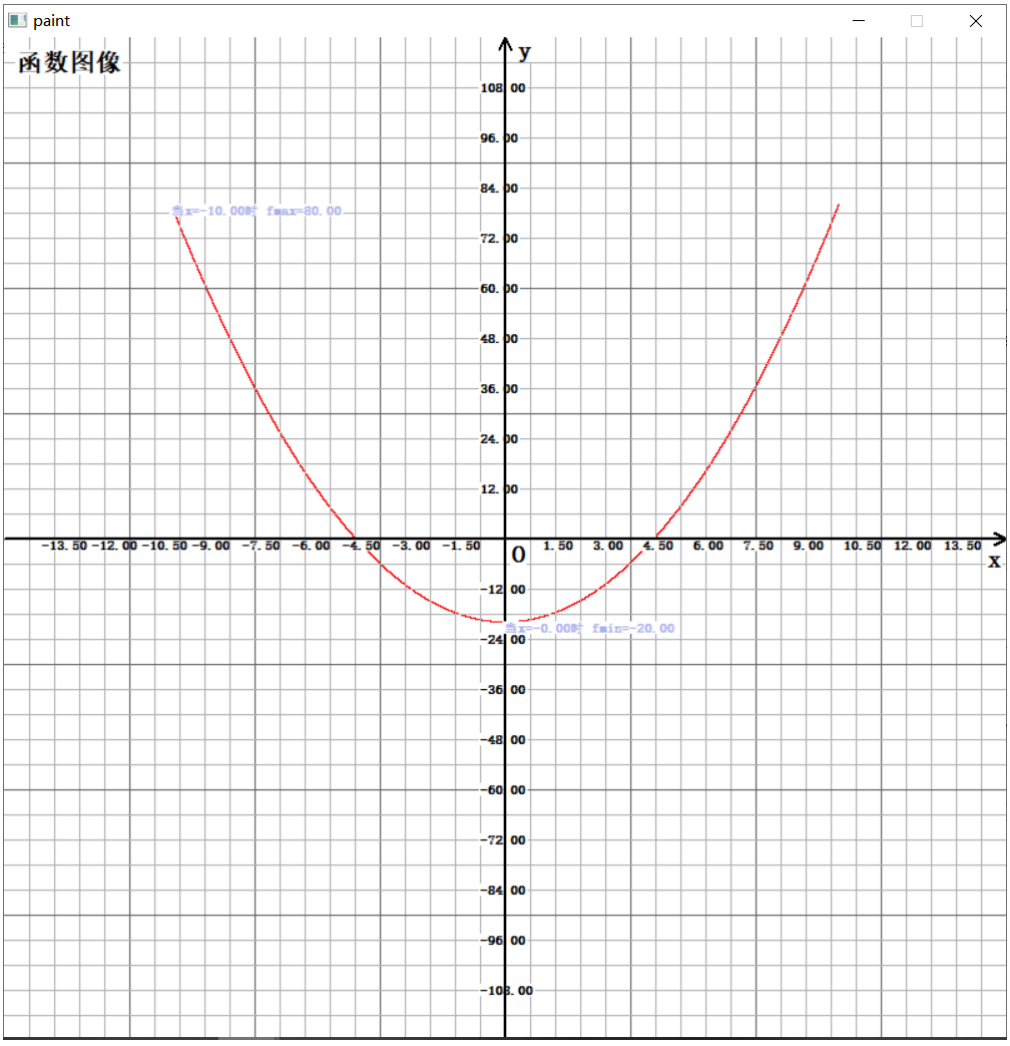
下面展示一些我的程序所绘制的函数图像。



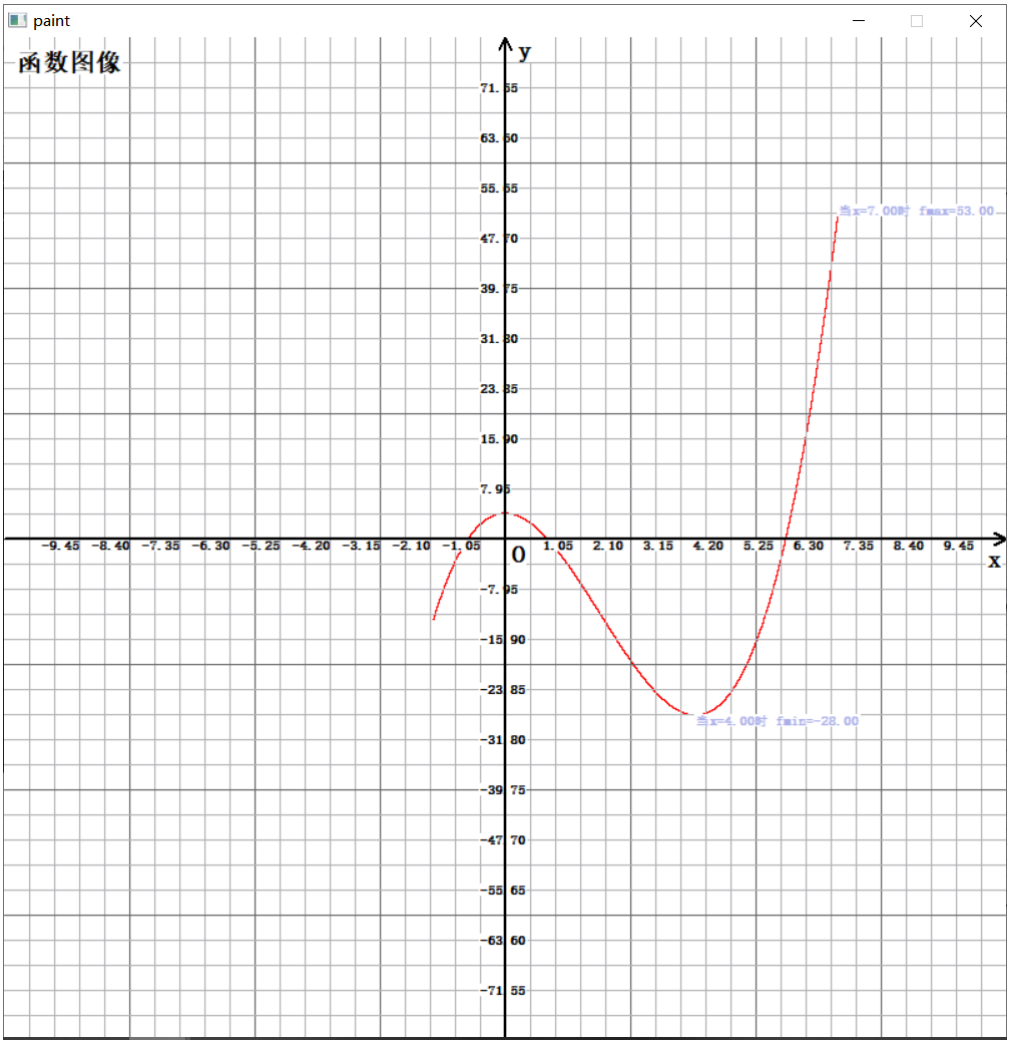

```
在这个程序中，你可以输入函数表达式和自变量区间，本程序会帮助你绘制函数图像
输入函数表达式（如x^2-2x+1）:x 0.5
输入坐标区间（如：-1.5 1.5）:0 10
```



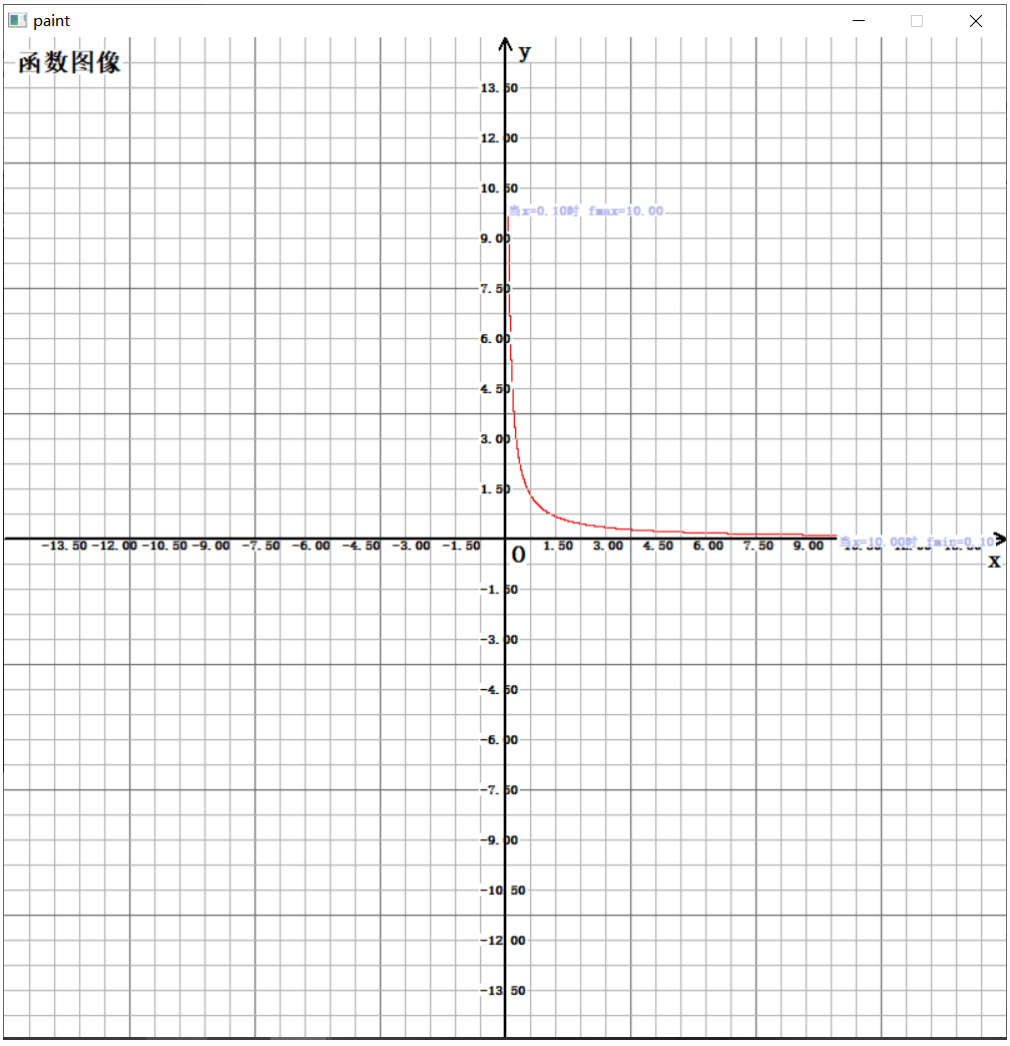

```
在这个程序中，你可以输入函数表达式和自变量区间，本程序会帮助你绘制函数图像
输入函数表达式（如x^2-2x+1):x^2-20
输入坐标区间（如：-1.5 1.5）:-10 10
```




```
在这个程序中，你可以输入函数表达式和自变量区间，本程序会帮助你绘制函数图像
输入函数表达式（如x^2-2x+1):x^3-6x^2+4
输入坐标区间（如：-1.5 1.5）:-1.5 7
```




```
在这个程序中，你可以输入函数表达式和自变量区间，本程序会帮助你绘制函数图像
输入函数表达式（如x^2-2x+1):x^-1
输入坐标区间（如：-1.5 1.5）:0.1 10
```



以上就是此次我编写的函数绘图程序所绘制的图像样例。

四、实验体会和心得

此次绘制函数图像的程序，与`P1`专题的`Dynamic Array`和`Linked List`作业相比较，综合性和应用性更强，对于前面所学到的知识都有一定程度的涉及。

我在写这个程序中的过程中，也发现了很多以我现有知识无法解决的问题，

读入`creatfunc()`部分，我在调试的过程中发现，这个读入多项式的函数部分存在一定的缺陷——无法读入首项系数是负数的函数表达式，当输入负数时便会出现`bug`，可能是因为开始时定义的`op`是`+`，导致首项的系数无法读入负数。这个部分也想到了一些解决的办法，例如逐项读入每一个字符并判断不同的情况。但在这个过程中，系数和项数的循环也会出现很多问题，例如判断数字是否为一位数或者是否为小数，而不同的情况对应的存储情况不同：当数字不为一位数时，需要不断循环得到各个位置的数字存储起来；当数字为浮点小数时，又需要判断`'.'`的出现，出现时又应如何应对。这些情况都有非常复杂的应对措施，而且在尝试的过程中总会不断发现不足之处，因此希望这个部分能在接下来的学习中有所进步，对这个方面可以有进一步的优化。

关于函数图像的绘制还有进一步可以优化之处。坐标轴的尺寸标度方面，我看到很多函数图像绘制软件的尺寸标度是不随着函数值的改变而改变的，但这个前提是其界面可以进行缩放，随着函数值的增大或减小而缩放，这样保证了在绘制函数的过程中，函数图像更加接近原本的情况。例如，在一次函数的绘制过程中，就不会出现图像中直线斜率不是`1`的情况了，但这也可能涉及基本库中无法实现的功能。除此之外，在某些情况下， 曲线的绘制会比较像直线，但这并不全是程序的问题，只是无法具体地体现斜率的变化。函数图像的绘制在本次程序中采取了“描点”的方法，但并没有将各个点顺次连接。我发现以“描点法”绘制的函数图像在某些情况下会出现“失真”的情况——曲线不光滑，但这一点也与其本

身有关，而且大多数情况下都是比较光滑的曲线。另外，我发现自己的程序的响应时间比较慢，但编译运行的时间比较多，这一点我并未找到原因，希望在后面的学习中能够不断优化。

在写这个程序的过程中我学到了一些新的知识，例如`scanf`的返回值，`sprintf`的用法等，也运用数学知识实现了两个坐标系的坐标变换，对链表的使用也加深了印象。在此之前，我关于这种画图的程序和项目的编写模式是比较陌生的，包括对`acllib`的各种函数也只停留在了解的阶段，无法熟练地掌握和运用各种函数进行画图。这次函数绘图的程序作业很好地帮我巩固了之前自学的知识，我也在写这个程序的过程中找到了很多快乐，怎么样把程序写得更好，让最终呈现的图形界面更加美观也是一件非常有趣的事情，而当最后的图像呈现在自己面前时，也是非常有成就感的。希望在日后的学习中能够更加熟练地掌握此方面的知识并能够对其他学科的学习有一定的帮助。