//　　附录 一些典型算法的程序实例
//
//　本附录中的所有源程序文件可通过 e-mail 向读者发送，如有任何问题、要求或建议，欢迎给作者发 e-mail，作者的 e-mail 地址是 bxjsyjs@hotmail.com。请读者注明 e-mail 接收地址。
//　以下各节中的程序基本上与本书前面各章节相对应，读者在学完相关章节后，做对应的上机实习程序。
//　为了保证每个程序的完整性，使得都能单独运行，下列各节中不同的程序之间有很多重复的代码和相同的 C 函数，尤其是一些输入/输出函数。这样安排的好处是读者可以任意选择一个范例程序，而不用阅读这一范例程序以外的其它内容。所以请读者在输入程序时注意，对相同的代码和 C 函数可直接在文件之间拷贝。
//
//　§1 一个用于数学函数值计算的 C 函数 — 求任意数学函数 f(x)和 f(x,y)的值
//
//
//　任意输入一个数学函数的表达式，在输入自变量的值后，计算出数学函数表达式的值这在很多情况下都会遇到。例如，一元函数 y=2sinx+1，求 x=5 时的函数值 y；或二元函数 z=3cos(x+1)+y,求 x=2,y=3 时的函数值 z；或三元函数 q=2x+3y+sin(z+2x), 求 x=1,y=2,z=3 时的函数值 q。
//　以下程序中的三个 C 函数 float f(float x) 、float f(float x,float y)和 float f(float x,float y,float z) 分别实现一元函数、二元函数和三元函数值的计算（因为这三个 C 函数的名字相同，都为 f，只是参数个数不同，这用到了 C++中函数重载的功能，所以要用 C++编绎，即源程序文件的扩展名应取 CPP）。
//　程序实现中用到了“编绎原理”中有关表达式编绎的知识，包括表达式的词法分析、语法分析和语义生成，有兴趣的读者请参阅有关书籍。
//　程序运行后，数学函数的输入格式采用 C 语言算术表达式的格式，例如对于一元函数 y=2sinx+1，则输入 2*sin(x)+1；对于二元函数 z=3cos(x+1)+y，则输入 3*cos(x+1)+y；对于三元函数 q=2x+3(y-2)+sin(z+x)，则输入 2*x+3*(y-2)+sin(z+2*x)等等。
//　需要提醒读者注意的是，本程序的语法分析功能极其有限，仅仅提供了一个语法分析的 C 函数接口，有兴趣的读者可以自己添加语法分析代码。所以对错误的 C 语言表达式输入，大多不能报错。因此，使用时务必输入正确的 C 语言表达式。
//　下列程序是实现 C 语言算术表达式计算的函数，文件名取为 expressi .cpp。使用方式有二种，一是加入到读者的 project 中；二是用 #include "expressi.cpp" 语句包含到读者的源程序文件中。本章的范例程序采用后者。所以，如果范例程序中有 #include "expressi.cpp" 语句，则读者应首先把以下程序输入到计算机中，并用文件名 expressi.cpp 存盘。
//
//
//计算 C 语言算术表达式程序，用于一元或多元函数值的计算
//

```
#include  <stdio.h>
#include  <conio.h>
#include  <math.h>
#include  <string.h>
```

```
#define    ADD       0xff01
#define    SUB 0xff02
#define    MUL       0xff03
#define    DIV 0xff04
#define LEFT_PARENTHESES    0xff05
#define RIGHT_PARENTHESES 0xff06
#define COMMA    0xff07

#define ADD1 0xff07
#define SUB1  0xff08
#define EQU   0xff09

#define    SIN  0xff10
#define    COS 0xff11
#define    TAN 0xff12
#define    ASIN    0xff13
#define ACOS 0xff14
#define ATAN 0xff15
#define EXP    0xff16
#define LOG   0xff17
#define POW  0xff18
#define SQRT 0xff19
#define FABS  0xff1a
#define FACTORIAL     0xff1b
#define MINUS      0xff1c

struct OPERATOR_FUNCTION_NAME_LIST
{
      char Name[32];
      int Code;
      int Pri;
      int NumOfOper;
}OF_Name[] =
{
      { "+",      0xff01, 1, 2 },
      { "-",      0xff02, 1, 2 },
      { "*",      0xff03, 2, 2 },
      { "/",      0xff04, 2, 2 },
      { "(",      0xff05, 4, 0 },
      { ")",      0xff06, 0, 0 },
      { ",",      0xff07, 0, 0 },

      { "sin",    0xff10, 3, 1 },
      { "cos",    0xff11, 3, 1 },
```

```
        { "tan",    0xff12, 3, 1 },
        { "asin",   0xff13, 3, 1 },
        { "acos",   0xff14, 3, 1 },
        { "atan",   0xff15, 3, 1 },
        { "exp",    0xff16, 3, 1 },
        { "log",    0xff17, 3, 1 },
        { "pow",    0xff18, 3, 2 },
        { "sqrt",   0xff19, 3, 1 },
        { "fabs",   0xff1a, 3, 1 },
        { "factorial",    0xff1b, 3, 1 },
        { "minus", 0xff1c, 5, 1 },
        { "", 0, 0 }
};

float Factorial(float n)
{
        float Result,ftmp;

        ftmp=n;
        Result=1.;
        while(ftmp>1.)
        {
                Result*=ftmp;
                ftmp-=1.;
        }
        return(Result);
}

int NextWordLen(char e[])
{
        int i;

        if(e[0]=='+'||e[0]=='-'||e[0]=='*'||e[0]=='/'||
                e[0]=='('||e[0]==')'||e[0]==',')
                return 1;
        else
        {
                i=0;
                do
                {
                        ++i;
                }while(e[i]!='+'&&e[i]!='-'&&e[i]!='*'&&e[i]!='/'&&
                                e[i]!='('&&e[i]!=')'&&e[i]!=','&&e[i]!='\0');
                return i;
```

```
        }
}

int Check_OF_Name_List(char Word[])
{
        int i;

        i=0;
        while(OF_Name[i].Name[0]!=0)
        {
                if(strcmp(Word,OF_Name[i].Name)==0)
                        return(OF_Name[i].Code);
                ++i;
        }

        return 1;
}

int CheckOrAddVarNameList(char Word[],char VarNameList[][32])
{
        int i;

        i=0;
        while(VarNameList[i][0]!=0)
        {
                if(strcmp(Word,VarNameList[i])==0)
                        return(i);
                ++i;
        }

        strcpy(VarNameList[i],Word);
        VarNameList[i+1][0]='\0';
        return(i);
}

int GetFormated_C_Expression(char C_Expression[],int Fmt_C_Exp[],
                                                char VarNameList[][32])
{
        int i,i1,j,WordLen;
        char Word[32];

        i=0;
        j=0;
        while(C_Expression[i]!=0)
```

```
    {
        WordLen=NextWordLen(&C_Expression[i]);
        strncpy(Word,&C_Expression[i],WordLen);
        Word[WordLen]='\0';

        i1=Check_OF_Name_List(Word);
        if(i1<0)
        {

            Fmt_C_Exp[j]=i1;
            if(i1==SUB&&(Fmt_C_Exp[j-1]==LEFT_PARENTHESES||Fmt_C_Exp[j-
1]==COMMA))
                Fmt_C_Exp[j]=MINUS;
        }
        else
            Fmt_C_Exp[j]=CheckOrAddVarNameList(Word,VarNameList);
        ++j;

        i+=WordLen;
    }
    Fmt_C_Exp[j]=0xffff;
    return 0;
}

int OperNum(int Code)
{
    int i;

    i=0;
    while(OF_Name[i].Code!=0)
    {
        if(Code==OF_Name[i].Code)
            return(OF_Name[i].NumOfOper);
        ++i;
    }
    return 0;
}

int IsValidExpression(int Fmt_C_Exp[],char VarNameList[][32])
{
    int i,Valid,Parentheses;

    Parentheses=0;
    Valid=0;
```

```
	i=0;
	while(Fmt_C_Exp[i]!=0xffff)
	{
		if(((Fmt_C_Exp[i]>=0xff01&&Fmt_C_Exp[i]<=0xff04)||Fmt_C_Exp[i]==MINUS)&&
				(((Fmt_C_Exp[i-1]>=0xff01&&Fmt_C_Exp[i-1]<=0xff04)||Fmt_C_Exp[i-
1]==MINUS)||
			((Fmt_C_Exp[i+1]>=0xff01&&Fmt_C_Exp[i+1]<=0xff04)||
Fmt_C_Exp[i+1]==MINUS)))
				Valid=1;
		if(Fmt_C_Exp[i]==LEFT_PARENTHESES)++Parentheses;
		if(Fmt_C_Exp[i]==RIGHT_PARENTHESES)--Parentheses;

		/*Other Invalid Case can be added here Or
		rewrite this function at all by user!,
		Otherwise,You must input a correct C_Expression!*/

		/***********************************/

		++i;
	}
	if(Parentheses!=0)Valid=1;
	return(Valid);
}

int Pri(int Code)
{
	int i;

	i=0;
	while(OF_Name[i].Code!=0)
	{
		if(Code==OF_Name[i].Code)
			return(OF_Name[i].Pri);
		++i;
	}
	return 0;
}

int GetOperatorSerials(int Fmt_C_Exp[],int OperatorSerials[][4],int VarP)
{
	int OperP;
	int i,j;
	int O_Stack[200],O_P;
```

```
int V_Stack[200],V_P;
int itmp1;

OperP=0;
O_P=0;
V_P=0;
i=0;
while(Fmt_C_Exp[i]!=0xffff)
{
    if(Fmt_C_Exp[i]<2000&&Fmt_C_Exp[i]>=0)
    {
        V_Stack[V_P]=Fmt_C_Exp[i];
        ++V_P;
    }
    else
    {
        if(Fmt_C_Exp[i]!=LEFT_PARENTHESES)
            while(O_P>0&&Pri(Fmt_C_Exp[i])<=Pri(O_Stack[O_P-1]))
            {
                if(O_Stack[O_P-1]==LEFT_PARENTHESES&&
                    Fmt_C_Exp[i]==RIGHT_PARENTHESES)
                {
                    --O_P;
                    break;
                }
                if(O_Stack[O_P-1]!=LEFT_PARENTHESES)
                {
                    switch(OperNum(O_Stack[O_P-1]))
                    {
                        case 0:
                            --O_P;
                            break;
                        case 1:
                            OperatorSerials[OperP][0]=O_Stack[O_P-1];
                            OperatorSerials[OperP][1]=V_Stack[V_P-1];
                            OperatorSerials[OperP][3]=VarP;
                            V_Stack[V_P-1]=VarP;
                            ++VarP;
                            ++OperP;
                            --O_P;
                            break;
                        case 2:
                            OperatorSerials[OperP][0]=O_Stack[O_P-1];
                            OperatorSerials[OperP][2]=V_Stack[V_P-1];
```

```
                                        OperatorSerials[OperP][1]=V_Stack[V_P-2];
                                        OperatorSerials[OperP][3]=VarP;
                                        V_Stack[V_P-2]=VarP;
                                        ++VarP;
                                        --V_P;
                                        ++OperP;
                                        --O_P;
                                        break;
                                }
                        }
                        else
                                break;
                }
                if(Fmt_C_Exp[i]!=RIGHT_PARENTHESES)
                {
                        O_Stack[O_P]=Fmt_C_Exp[i];
                        ++O_P;
                }
        }
        ++i;
    }
    OperatorSerials[OperP][0]=0;
    return 0;
}

int VarNameListLen(char VarNameList[][32])
{
    int Length;

    Length=0;
    while(VarNameList[Length][0]!='\0')++Length;
    return(Length);
}

int MakeFunction(char C_Expression[],int OperatorSerials[][4],
                        char VarNameList[][32])
{
    int Fmt_C_Exp[1000];

    GetFormated_C_Expression(C_Expression,Fmt_C_Exp,VarNameList);
    if(IsValidExpression(Fmt_C_Exp,VarNameList))return 1;

    GetOperatorSerials(Fmt_C_Exp,OperatorSerials,VarNameListLen(VarNameList));
```

```
	return 0;
}

void SetVarInitValue(char VarNameList[][32],float VarsSpace[])
{
	int i;

	i=0;
	while(VarNameList[i][0]!='\0')
	{
		if(VarNameList[i][0]>='0'&&VarNameList[i][0]<='9')
		  VarsSpace[i]=atof(VarNameList[i]);
		else
		{
//============================================================
//			printf("\nInput: %s=",VarNameList[i]);
//			scanf("%f",&VarsSpace[i]);
//============================================================*/
		}
		++i;
	}
}

float CalculationOfSerials(int OperatorSerials[][4],float VarsSpace[])
{
	int i;
	float a,b;

	i=0;
	while(OperatorSerials[i][0]!=0)
	{
		a=VarsSpace[OperatorSerials[i][1]];
		b=VarsSpace[OperatorSerials[i][2]];
		switch(OperatorSerials[i][0])
		{
			case EQU:
				VarsSpace[OperatorSerials[i][3]]=a;
				break;
			case ADD:
				VarsSpace[OperatorSerials[i][3]]=a+b;
				break;
			case ADD1:
				VarsSpace[OperatorSerials[i][3]]=+a;
				break;
```

```
case SUB:
    VarsSpace[OperatorSerials[i][3]]=a-b;
    break;
case SUB1:
    VarsSpace[OperatorSerials[i][3]]=-a;
    break;
case MUL:
    VarsSpace[OperatorSerials[i][3]]=a*b;
    break;
case DIV:
    VarsSpace[OperatorSerials[i][3]]=a/b;
    break;
case SIN:
    VarsSpace[OperatorSerials[i][3]]=sin(a);
    break;
case COS:
    VarsSpace[OperatorSerials[i][3]]=cos(a);
    break;
case TAN:
    VarsSpace[OperatorSerials[i][3]]=tan(a);
    break;
case ASIN:
    VarsSpace[OperatorSerials[i][3]]=asin(a);
    break;
case ACOS:
    VarsSpace[OperatorSerials[i][3]]=acos(a);
    break;
case ATAN:
    VarsSpace[OperatorSerials[i][3]]=atan(a);
    break;
case EXP:
    VarsSpace[OperatorSerials[i][3]]=exp(a);
    break;
case LOG:
    VarsSpace[OperatorSerials[i][3]]=log(a);
    break;
case POW:
    VarsSpace[OperatorSerials[i][3]]=pow(a,b);
    break;
case SQRT:
    VarsSpace[OperatorSerials[i][3]]=sqrt(a);
    break;
case FABS:
    VarsSpace[OperatorSerials[i][3]]=fabs(a);
```

```
                    break;
                case FACTORIAL:
                    VarsSpace[OperatorSerials[i][3]]=Factorial(a);
                    break;
                case MINUS:
                    VarsSpace[OperatorSerials[i][3]]=-a;
                    break;

            }
            ++i;
        }
        /*The value calculated by the last step is the return result.*/
        return(VarsSpace[OperatorSerials[i-1][3]]);
}

void InputFx(char String[])
{
        printf("\nInput Function(with Varible x): ");
        scanf("%s",&String[1]);
        String[0]='(';
        strcat(String,")");
}

//================================================
int fx_OperatorSerials[200][4];
float fx_VarsSpace[200];
char fx_VarNameList[200][32]={"x","\0"};
char fx_C_Expression[1000];


int CreateFx(char String[])
{
        strcpy(fx_VarNameList[0],"x");
        strcpy(fx_VarNameList[1],"\0");

        strcpy(&fx_C_Expression[1],String);
        fx_C_Expression[0]='(';
        strcat(fx_C_Expression,")");

        if(MakeFunction(fx_C_Expression,fx_OperatorSerials,fx_VarNameList))
            {
                printf("\nExpression Wrong!");
                return 1;
            }
```

```
    SetVarInitValue(fx_VarNameList,fx_VarsSpace);
    return 0;
}

int CreateFxy(char String[])
{
    strcpy(fx_VarNameList[0],"x");
    strcpy(fx_VarNameList[1],"y");
    strcpy(fx_VarNameList[2],"\0");

    strcpy(&fx_C_Expression[1],String);
    fx_C_Expression[0]='(';
    strcat(fx_C_Expression,")");

    if(MakeFunction(fx_C_Expression,fx_OperatorSerials,fx_VarNameList))
        {
            printf("\nExpression Wrong!");
            return 1;
        }
    SetVarInitValue(fx_VarNameList,fx_VarsSpace);
    return 0;
}

float f(float x)
{
    fx_VarsSpace[0]=x;   //if single compute then rem this sentence
    return(CalculationOfSerials(fx_OperatorSerials,fx_VarsSpace));
}

float f(float x,float y)
{
    fx_VarsSpace[0]=x;   //if single compute then rem this sentence
    fx_VarsSpace[1]=y;
    return(CalculationOfSerials(fx_OperatorSerials,fx_VarsSpace));
}


//程序结束
////////////////////////////////////////////////////////////////////////////////////////////////
```