Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4
« К а л ь к у л я т о р »

По дисциплине «Методы программирования»

Выполнили А.Э.Палёный
Студент гр. 13508/13 А.Романов
_____

Проверил
Преподаватель
_____ В.Б.Вагисаров

Санкт-Петербург
2016

Цель

Создать калькулятор, работающий с операциями – «+,-,/,*,^» и функциями «sin(),cos(),sqrt(),log() », а также с функциями, заданными собственноручно.

Ход работы

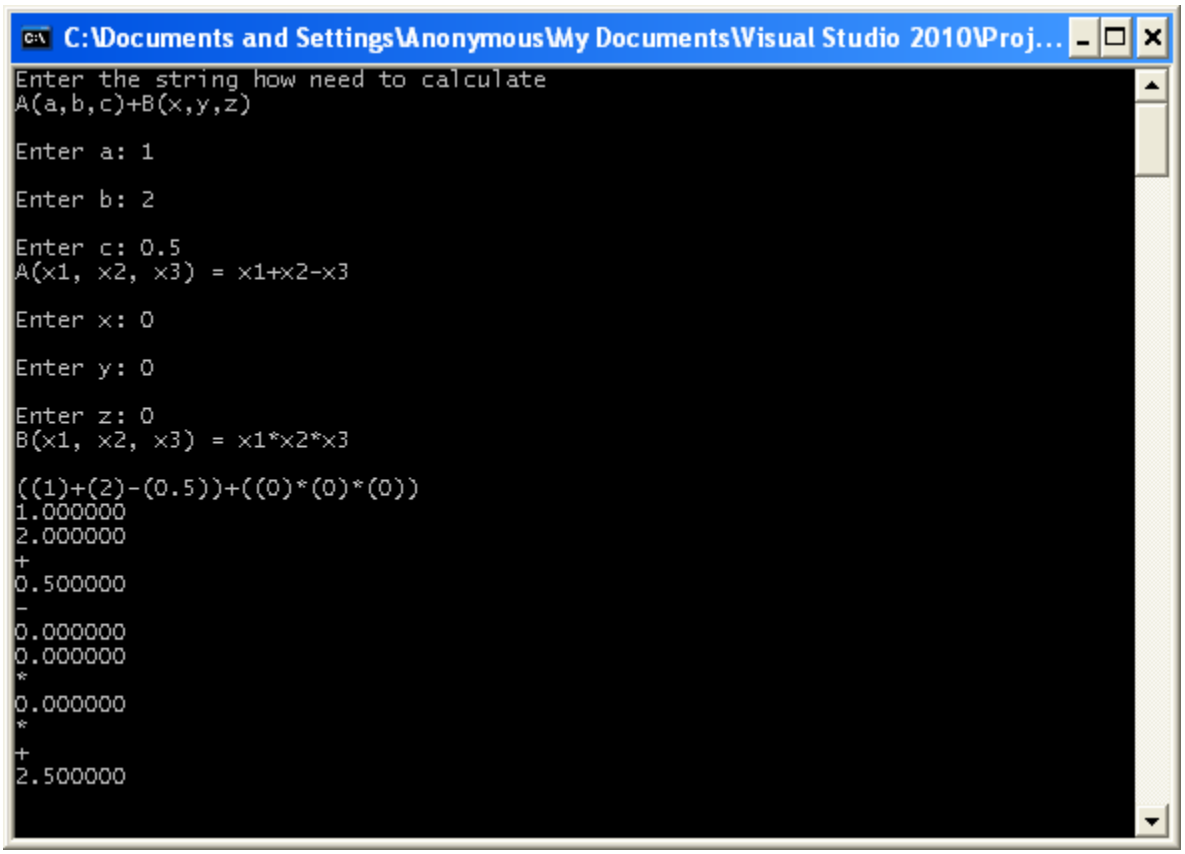
Была создана программа калькулятор, которая состоит из двух частей:

1) В данной части происходит преобразование выражения, для следующей части, в которой это выражение считается. В обязанности это части входит:
    a. Считывание первоначальной строки и выявление переменных функций и ошибок
    b. Запрос значений для переменных и подстановка этих значений на места переменных
    c. Запрос значений и выражения для функции

Дальше полученная строка передаётся во вторую часть.

2) Тут происходит преобразование выражения в линейный список или в так называемую «обратную польскую запись».
    После данного преобразования программа считывает значение и кладёт эго в стек. Но если программа считывает операцию, то значение забирается из буфера, затем считается, а затем кладётся в буфер.

3) Пример работы программы:



Исходный код

«IsThisErrosVariablesFunctions.h»

```
#include "Main.h"

//----------Main function----------------
int IsThisErrosVariablesFunctions(char **str, int length);

//-------------Service-------------------
int ReadVariable(char *str, Replace **Head, char **NewStr, int *LenNewStr);                    //Read variable func
int ReadFunction(char *name, char **bone, int iCountVar, ReplFunk **Head, char **NewStr, int *LenNewStr, int Hook);
int IfIKnow(char *str, int len, int *i, char **NewStr, int *LenNewStr, Replace **Head);        //Repair If i know this
functions(sin,cos,log,sqrt)
int Obschit(char *str, int *i, Replace **Head, ReplFunk **Body, char *NewStr, int *LenNewStr, int len);
```

«IsThisErrosVariablesFunctions.c»

```
#include "IsThisErrosVariablesFunctions.h"
#include <stdio.h>
#include <math.h>
```

```
#include <malloc.h>
#include <string.h>
#include <stdlib.h>

//-------------------------------------------------------------------------------
//<<<<<<<<<<<<<<<<<<<<<<----------Realization----------->>>>>>>>>>>>>>>>>>>>>>>>>
//_____________________________________________________________________________________

int IsThisErrosVariablesFunctions(char **str, int length)
{
	int LenNewStr=1;
	int i;
	char *NewStr=(char*)malloc(LenNewStr+1);
	Replace *Head=(Replace*)malloc(sizeof(Replace));
	ReplFunk *Body=(ReplFunk*)malloc(sizeof(ReplFunk));

	NewStr[0]=NULL;

	Body->iCountVar=NULL;
	Body->next=NULL;
	Body->past=NULL;
	Body->str=NULL;
	Body->name=NULL;

	Head->from=NULL;
	Head->next=NULL;
	Head->past=NULL;
	Head->to=NULL;

	for(i=0; i<length; i++, LenNewStr++)
	{
		if(!IfIKnow(str[0], length, &i, &NewStr, &LenNewStr, &Head, &Body))	return 0;
		NewStr=(char*)realloc(NewStr, LenNewStr+1);
		NewStr[LenNewStr-1]=str[0][i];
		NewStr[LenNewStr]=0;
	}
	free(str[0]);
	DestrReplace(&Head);
	DestrReplFunk(&Body);
	str[0]=NewStr;
}

int ReadVariable(char *str, Replace **Head, char **NewStr, int *LenNewStr)
{
//Read only Variables
```

```
	char *dump;
	int len=0;
	int i;
	Replace **Dump=Head;

	for(i=0; str[i]!=NULL; i++)
	{
		if(!IsNumber)		return 1;
	}

	if(ReadToReplace(Dump, str))
	{
		*LenNewStr+=strlen(Dump[0]->to);
		*NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
		strcat(*NewStr, Dump[0]->to);
		return 0;
	}

	printf("%s = ",str);
	len=Entr(&dump);

	//Need to repair user's trash
	AddToReplace(Head, str, dump);

	*LenNewStr+=strlen(dump);
	*NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
	if(NewStr[0][0]==NULL)	strcpy(*NewStr, dump);
	else	strcat(*NewStr, dump);
	return 0;
	//Need to repair on repeat
}

int ReadFunction(char *name, char **bone, int iCountVar, ReplFunk **Head, char **NewStr, int *LenNewStr, int Hook)
{
//Read only Functions
	char *dump;
	int len=0,pok=0;
	ReplFunk **Dump=Head;
	int i;
	char *str;
	char *count;

	*NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
	NewStr[0][*LenNewStr-1]='(';
	NewStr[0][*LenNewStr]=0;
```

```
*LenNewStr+=1;

if(ReadToReplFunk(Dump, name, iCountVar))
{
	for(i=0; Head[0]->str[i]!=NULL; i++, *LenNewStr+=1)
	{
		if(Head[0]->str[i]=='x')
		{
			i++;
			count=(char*)malloc(1);
			for(len=0;IsNumber(Dump[0]->str[i]);i++, len++)
			{
				count=(char*)realloc(count, len+2);
				count[len]=Dump[0]->str[i];
				count[len+1]=0;
			}
			if(!len)
			{
				*LenNewStr+=1;
				for(len=0; bone[0][len]!=NULL; len++, *LenNewStr+=1)
				{
					*NewStr=(char*)realloc(*NewStr,*LenNewStr+1);
					NewStr[0][*LenNewStr-1]=bone[0][len];
					NewStr[0][*LenNewStr]=0;
				}
			}else{
				pok=atoi(count);
				free(count);
				if(pok<=0 || pok>iCountVar) return 1;
				//*LenNewStr+=1;
				*NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
				NewStr[0][*LenNewStr-1]='(';
				NewStr[0][*LenNewStr]=0;
				*LenNewStr+=1;
				for(len=0; bone[pok-1][len]!=NULL; len++, *LenNewStr+=1)
				{
					*NewStr=(char*)realloc(*NewStr,*LenNewStr+1);
					NewStr[0][*LenNewStr-1]=bone[pok-1][len];
					NewStr[0][*LenNewStr]=0;
				}
				*NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
				NewStr[0][*LenNewStr-1]=')';
				NewStr[0][*LenNewStr]=0;
				*LenNewStr+=1;
				for(len=0; len<Hook; len++)
```

```
                        {
                                *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
                                NewStr[0][*LenNewStr-1]=')';
                                NewStr[0][*LenNewStr]=0;
                                *LenNewStr+=1;
                        }
                }
                len=0;
        }
        if(Head[0]->str[i]!=NULL)
        {
                *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
                NewStr[0][*LenNewStr-1]=Head[0]->str[i];
                NewStr[0][*LenNewStr]=0;
        }else{  i-=1;   *LenNewStr-=1;}
    }
    *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
    NewStr[0][*LenNewStr-1]=')';
    NewStr[0][*LenNewStr]=0;
    *LenNewStr+=1;
    return 0;
}

printf("%s(",name);
for(len=0; len<iCountVar; len++)
{
    printf("x%d, ",len+1);
}
printf("\b\b) = ");
len=Entr(&str);


//Need to repair user's trash
AddToReplFunk(Head,name,str,iCountVar);

    for(i=0; Head[0]->str[i]!=NULL; i++, *LenNewStr+=1)
    {
        if(Head[0]->str[i]=='x')
        {
            i++;
            count=(char*)malloc(1);
            for(len=0;IsNumber(str[i]);i++, len++)
            {
                count=(char*)realloc(count, len+2);
                count[len]=str[i];
```

```
                count[len+1]=0;
        }
        if(!len)
        {
                *LenNewStr+=1;
                for(len=0; bone[0][len]!=NULL; len++, *LenNewStr+=1)
                {
                        *NewStr=(char*)realloc(*NewStr,*LenNewStr+1);
                        NewStr[0][*LenNewStr-1]=bone[0][len];
                        NewStr[0][*LenNewStr]=0;
                }
        }else{
                pok=atoi(count);
                free(count);
                if(pok<=0 || pok>iCountVar) return 1;
                //*LenNewStr+=1;
                *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
                NewStr[0][*LenNewStr-1]='(';
                NewStr[0][*LenNewStr]=0;
                *LenNewStr+=1;
                for(len=0; bone[pok-1][len]!=NULL; len++, *LenNewStr+=1)
                {
                        *NewStr=(char*)realloc(*NewStr,*LenNewStr+1);
                        NewStr[0][*LenNewStr-1]=bone[pok-1][len];
                        NewStr[0][*LenNewStr]=0;
                }
                *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
                NewStr[0][*LenNewStr-1]=')';
                NewStr[0][*LenNewStr]=0;
                *LenNewStr+=1;
                for(len=0; len<Hook; len++)
                {
                        *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
                        NewStr[0][*LenNewStr-1]=')';
                        NewStr[0][*LenNewStr]=0;
                        *LenNewStr+=1;
                }
        }
        len=0;
}

if(Head[0]->str[i]!=NULL)
{
        *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
        NewStr[0][*LenNewStr-1]=Head[0]->str[i];
```

```
                                        NewStr[0][*LenNewStr]=0;
                                }else{      i-=1;       *LenNewStr-=1;}
//                              *LenNewStr+=1;
                        }
                        *NewStr=(char*)realloc(*NewStr, *LenNewStr+1);
                        NewStr[0][*LenNewStr-1]=')';
                        NewStr[0][*LenNewStr]=0;
                        *LenNewStr+=1;
                        return 0;
                //Need to repair on repeat
}

int IfIKnow(char *str, int len, int *i, char **NewStr, int *LenNewStr, Replace **Head, ReplFunk **Body)
{
                char *dump;
                int SinSqrt=0;
                int k,m;
                int IsEnded=0;
                int DumpedHooks=0;

                for(; str[*i]!=0; *i+=1)
                {
                        switch (str[*i])
                        {
                        case '(':
                                {
                                        if(str[*i+1]==')' || (!IsZnak(str[*i-1]) && str[*i]!='('))          return 0;
                                        else                                        return 1;
                                }
                        case ')':
                                {
                                        if(str[*i+1]==')' || (!IsZnak(str[*i-1]) && str[*i]!=')'))          return 0;
                                        else                                        return 1;
                                }
                        case 'c':
                                {
                                        if(!RepairCos(str, i, NULL, 0, DumpedHooks))
                                        {
                                                IsEnded=Obschit(str, i, Head, Body, NewStr, LenNewStr,len);

                                                if(IsEnded)                 return 0;
                                                else                        return 1;
                                        }else{
                                                dump=(char*)malloc(4);
                                                dump[0]=str[*i-3];
```

```
                        dump[1]=str[*i-2];
                        dump[2]=str[*i-1];
                        dump[3]=0;
                        *LenNewStr+=3;
                        *NewStr=(char*)realloc(*NewStr,*LenNewStr+1);
                        strcat(NewStr[0], dump);
                        return 1;
                }
        }
case 's':
        {
                SinSqrt=RepairSinSqrt(str, i, NULL, 0, DumpedHooks);
                if(!SinSqrt)
                {
                        IsEnded=Obschit(str, i, Head, Body, NewStr, LenNewStr,len);
                        if(IsEnded)                     return 0;
                        else                            return 1;
                }else{
                        SinSqrt+=2;
                        dump=(char*)malloc(SinSqrt);
                        SinSqrt--;
                        *LenNewStr+=SinSqrt;
                        for(k=0; SinSqrt>0; k++,SinSqrt--)
                                dump[k]=str[*i-SinSqrt];

                        dump[k]=0;
                        *NewStr=(char*)realloc(*NewStr,*LenNewStr+1);
                        strcat(NewStr[0], dump);
                        return 1;
                }
        }
case 'l':
        {
                if(!RepairLog(str, i, NULL, 0, DumpedHooks))
                {
                        IsEnded=Obschit(str, i, Head, Body, NewStr, LenNewStr,len);
                        if(IsEnded)                     return 0;
                        else                            return 1;
                }else{
                        dump=(char*)malloc(4);
                        dump[0]=str[*i-3];
                        dump[1]=str[*i-2];
                        dump[2]=str[*i-1];
                        dump[3]=0;
                        *LenNewStr+=3;
```

```
                                        *NewStr=(char*)realloc(*NewStr,*LenNewStr+1);
                                        strcat(NewStr[0], dump);
                                        return 1;
                                }
                        }
                default:
                        {
                                if(IsAlphabet(str[*i]))
                                {
                                        IsEnded=Obschit(str, i, Head, Body, NewStr, LenNewStr,len);
                                        if(IsEnded)                     return 0;
                                        else                            return 1;
                                }//IsAlphabet(str[*i])
                                if(!IsZnak(str[*i]) && !IsNumber(str[*i]) && str[*i]!='.')
                                return 0;
                        }//default
                }//switch
                return 1;
        }
}

int Obschit(char *str, int *i, Replace **Head, ReplFunk **Body, char **NewStr, int *LenNewStr, int len)
{
        int UnkLen=1;
        char *Unknown;
        int Hooks=0;
        int iHook=0;
        char **bone;
        int k,m,iReposM;
        char *var;
        int iVar=0,iWrite;
        int IsEnded=0;
        int LocalHook=0;

        Unknown=(char*)malloc(UnkLen+1);
        for(;*i<len && !IsZnak(str[*i]) && str[*i]!='(';*i+=1, UnkLen++)
        {
                Unknown=(char*)realloc(Unknown,UnkLen+1);
                Unknown[UnkLen-1]=str[*i];
                Unknown[UnkLen]=0;
        }

        if(str[*i]=='(')        Hooks++;
        if(Hooks)
        {
```

```
*i+=Hooks;

bone=(char**)malloc(sizeof(char*));
for(k=1; *i<len && str[*i]!=')'; *i+=1, k++)
{
	bone=(char**)realloc(bone,sizeof(char*)*k);
	bone[k-1]=(char*)malloc(sizeof(char));
	LocalHook=1;
	for(m=1, iHook=0; *i<len && (str[*i]!=')' || LocalHook!=1) && str[*i]!=','; *i+=1, m++)
	{
		if(IsAlphabet(str[*i]))
		{
			var=(char*)malloc(1);
			for(iVar=1;!IsZnak(str[*i]) && str[*i]!=')' && str[*i]!=','; *i+=1, iVar++)
			{
				var=(char*)realloc(var, sizeof(char)*(iVar+1));
				var[iVar-1]=str[*i];
				var[iVar]=0;
			}
			*i-=1;
			iReposM=m+iVar-1;
			printf("\nEnter %s: ",var);
			free(var);
			iVar=Entr(&var);

			iWrite=m+iVar;
			for(iVar=0; iWrite>m; m++, iVar++)
			{
				bone[k-1]=(char*)realloc(bone[k-1], sizeof(char)*(m+1));
				bone[k-1][m-1]=var[iVar];
				bone[k-1][m]=0;
			}
			m--;
			free(var);
			//m=iReposM;
			continue;
		}


		bone[k-1]=(char*)realloc(bone[k-1], sizeof(char)*(m+1));
		bone[k-1][m-1]=str[*i];
		bone[k-1][m]=0;
```

```
                        if(!IsNumber(str[*i]) && !IsZnak(str[*i]) && str[*i]!=')' && str[*i]!='('        return
1;
                        if(str[*i]=='('){       iHook++;LocalHook++;}
                        if(str[*i]==')')                                        LocalHook--;
                        switch (str[*i])
                        {
                        case '(':   if(str[*i+1]==')' || (!IsZnak(str[*i-1]) && str[*i]!='('))        return 1;break;
                        case ')':   if(str[*i+1]=='(' || (!IsZnak(str[*i-1]) && str[*i]!=')'))        return 1;break;
                        }
                }
                if(LocalHook!=1)   return 1;
                if(str[*i]==')')        *i-=1;
        }
    }

    if(Hooks)
    {
        k--;
        IsEnded=ReadFunction(Unknown, bone, k, Body, NewStr, LenNewStr, iHook);
        *i+=1+iHook;
        for(;k>0; k--)
        {
            free(bone[k-1]);
        }
        free(bone);
    }
    else    IsEnded=ReadVariable(Unknown, Head, NewStr, LenNewStr);
    if(IsEnded)         return 1;
    else                return 0;
}
```

«Main.h»

```
struct Zam
{
    struct Zam *next;
    struct Zam *past;
    char *from;
    char *to;
}typedef Replace;

struct Func
{
    struct Func *next;
    struct Func *past;
```

```
	int iCountVar;
	char *str;
	char *name;
}typedef ReplFunk;

struct elem
{
	struct elem *next;
	struct elem *past;
	double value;
	char operation;
}typedef Element;

struct prior
{
	struct prior *next;
	struct prior *past;
	char cSymbol;
	int iLevel;
}typedef Priority;

struct calculate
{
	struct calculate *next;
	struct calculate *past;
	double value;
}typedef Contar;

//---------------Procedures----------------

//Adding to structures
void AddToReplFunk(ReplFunk **Top, char *name, char *str, int iCountVar);	//To ReplFunk
void AddToReplace(Replace **Top, char *From, char *To);	//To
Replace
void AddToStack(Contar **Top, double Value);	//To
Contar
void AddToDump(Priority **Top, char cSymbol, int Level);	//To Priority
void Add(Element **Top, char operation, double value);	//To Element

//Reading
char ReadFromDump(Priority **Top);	//To Priority
int Movement(Element **Head);	//To
Element
int ReadToReplace(Replace **Top, char *str);	//To Replace
int ReadToReplFunk(ReplFunk **Top, char *name, int iCountVar); //To ReplFunk
```

```
//Destructors
void DestrReplace(Replace **Top);                                   //To Replace
void DestrReplFunk(ReplFunk **Top);                                 //To
ReplFunk
void DestrPriority(Priority **Top);                                 //To Priority
void DestrContar(Contar **Top);                                     //To
Contar
void DestrElement(Element **Top);                                   //To Element

//The main functions who used in the Repair:
int Skobki(char *str, int i, int len, int *Hooks, Priority **Dump, Priority *Nachalo, Element **Head);
int Znaki(char *str, int i, int len, int IsMinus);
int RepairCos(char *str, int *i, Priority **Dump, int write, int *Hooks);
int RepairLog(char *str, int *i, Priority **Dump, int write, int *Hooks);
int RepairSinSqrt(char *str, int *i, Priority **Dump, int write, int *Hooks);
void Calculate(char *str, Priority **Dump, Priority *Nachalo, int *i, int len, Element **Head, int *Hooks);
double ReadInt(char *str, int *i, int len);

//<<<<<<<<<<<<<<<Repair function>>>>>>>>>>>>>>>>>>
char Repair(char *str, int len, Element *Head);
//------------Read result----------------
int NeedToDo(Contar **Result, char operation);

//-----------Service--------------------
//Read all trash who was entered by user
int Entr(char **str);
//Repair if
int IsZnak(char str);
int IsNumber(char str);
int IsAlphabet(char str);        //Is this a symbol from alphabet

//-----------------Beta development--------------
//' ' - Space
int NotSpaceLeft(char *str, int *i, int IsMinus);
int NotSpaceRight(char *str, int *i, int len, int *iHooks);
```

«Main.c»

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include "Main.h"
#include <Windows.h>
```

```
//#include "vld.h"

#define TRUE 1
#define FALSE 0

int main()
{
	char *str;
	char dump=NULL;
	int length;
	double Value;
	Contar *Result;

	Element *Head;
	Element *Del;

	printf("Enter the string how need to calculate\n");
	length=Entr(&str);

	Result=(Contar*)malloc(sizeof(Contar));
	Head=(Element*)malloc(sizeof(Element));

	Head->next=NULL;
	Head->past=NULL;
	Head->operation=NULL;
	Head->value=0.0;

	Result->next=NULL;
	Result->past=NULL;
	Result->value=0.0;

	if(IsThisErrosVariablesFunctions(&str, length))
		printf("\n%s",str);
	else
	{
		printf("\n%s",str);
		printf("\nThere are errors");
		getchar();
		free(str);
		return 0;
	}

	length=strlen(str);
	Repair(str, length, Head);
	Del=Head;
```

```
	while(Movement(&Head))
	{
		if(Head->operation)
		{
			dump=Head->value;
			printf("\n%c",dump);
			if(!NeedToDo(&Result, &Head, dump))
			{
				printf("\nError");
				free(str);
				getchar();
				DestrElement(&Del);
				DestrContar(&Result);
				return 0;
			}
		}else{
			Value=Head->value;
			printf("\n%lf",Value);
			AddToStack(&Result, Value);
		}
	}
	printf("\n%lf",Result->value);

	DestrElement(&Del);
	DestrContar(&Result);
	free(str);
	getchar();
}

//Adding to structures
void AddToReplFunk(ReplFunk **Top, char *name, char *str, int iCountVar)
{
	ReplFunk *Next;
	Next=*Top;
	Next->next = (ReplFunk*)malloc(sizeof(ReplFunk));
	*Top=Next->next;
	Next->next->past=Next;
	Next->next->next=NULL;
	Next->next->iCountVar=iCountVar;
	Next->next->name=name;
	Next->next->str=str;
}

void AddToReplace(Replace **Top, char *From, char *To)
{
```

```
	Replace *Next;
	Next=*Top;
	Next->next = (Replace*)malloc(sizeof(Replace));
	*Top=Next->next;
	Next->next->past=Next;
	Next->next->next=NULL;
	Next->next->from=From;
	Next->next->to=To;
}

void AddToStack(Contar **Top, double Value)
{
	Contar *Next;
	Next=*Top;
	Next->next = (Contar*)malloc(sizeof(Contar));
	*Top=Next->next;
	Next->next->past=Next;
	Next->next->next=NULL;
	Next->next->value=Value;
}

void AddToDump(Priority **Top, char cSymbol, int Level)
{
	Priority *Next;
	Next=*Top;
	Next->next = (Priority*)malloc(sizeof(Priority));
	*Top=Next->next;
	Next->next->past=Next;
	Next->next->next=NULL;
	Next->next->iLevel=Level;
	Next->next->cSymbol=cSymbol;
}

void Add(Element **Top, char operation, double value)
{
	Element *Next;
	Next=*Top;
	Next->next = (Element*)malloc(sizeof(Element));
	*Top=Next->next;
	Next->next->past=Next;
	Next->next->next=NULL;
	Next->next->operation=operation;
	Next->next->value=value;
}
```

```
//Reading
char ReadFromDump(Priority **Top)
{
	Priority *Next=*Top;
	char c=Next->cSymbol;
	*Top=Next->past;
	free(Next);

	Next=*Top;
	Next->next=NULL;
	return c;
}

//Read all trash who was entered by user
int Entr(char **str)
{
	int i=0;
	char ch=0;
	*str=(char*)malloc(sizeof(char));

	for(i=1; ch = getchar(); i++)
	{
		if(ch!='\n')
		{
			*str=(char*)realloc(*str,sizeof(char)*(i+1));
			str[0][i-1]=ch;
		}else
			if(i!=1)	break;
			else i--;
	}
	i--;
	str[0][i]=0;

	return i;
}

//Beta development - ' '
int NotSpaceLeft(char *str, int *i, int IsMinus)
{
	if(IsMinus && IsZnak(str[*i+1]) && IsZnak(str[*i+2]))	return 1;
	if(!IsMinus && IsZnak(str[*i+1]))	return 1;
	return 0;
}

int NotSpaceRight(char *str, int *i, int len, int *iHooks)
```

```
{
	*i+=1;
	for(; *i<len ; *i+=1)
	{
		if(str[*i]=='(')	*iHooks+=1;
		if(str[*i]==')')	*iHooks-=1;
		if(str[*i]!=' ' && str[*i]!='(' && str[*i]!=')')	return 0; //you can go
	}
	return 1;
}

//The main functions who used in the Repair:
int Skobki(char *str, int *i, int len, int *Hooks, Priority **Dump, Priority *Nachalo, Element **Head)
{
	int IsMinus=0;

	//if(str[i]=='(' && str[i]==')')  return 1;
	if(str[*i]=='(')
	{
		*Hooks+=1;
		if(str[*i+1]=='-')	Add(Head, FALSE, ReadInt(str, i, len));
		else				if(NotSpaceRight(str, i, len, Hooks))	return 1;

		//if(i!=0){		if(NotSpaceLeft(str, &i, IsMinus));	return 1;}
		if(str[*i] > '/' && str[*i] < ':'){ *i-=1;	return 0;}//if cifra
	}

	if(str[*i]==')')
	{
		if(Dump[0]->past!=NULL && Dump[0]->iLevel==*Hooks)
			Add(Head, TRUE, ReadFromDump(Dump));

		*Hooks-=1;
		if(Dump[0]->past!=NULL && Dump[0]->iLevel==*Hooks)
			Add(Head, TRUE, ReadFromDump(Dump));
	}
	return 0;
}

int Znaki(char *str, int *i, int len, int IsMinus, Element **Head)
{
	int OurPlace=*i;
	int Hooks=0;

	if((*i+1)==len || *i==0 && IsMinus==FALSE)	return 1;
```

```
	//Left part
	if(NotSpaceLeft(str, i, IsMinus))		return 1;
	if((IsMinus && str[*i]=='(') || (IsMinus && *i==-1))		return 0;
	if(IsMinus)
	{
		//if((str[*i] < ')') || (str[*i] > '+' && str[*i] < '-') || (str[*i] > '-' && str[*i] < '/') || (str[*i] > '9'))		return
1;
		if(!IsAlphabet(str[*i+1]) && !IsNumber(str[*i+1]) && str[*i+1]!='(' && str[*i+1]!=')') return 1;

	//		if(IsZnak(str[*i-1]))		Add(Head, FALSE, ReadInt(str, i, len));
		//else				Add(Head, TRUE, str[*i]);
	}
	else
	{
		//if((str[*i] < ')') || (str[*i] > ')' && str[*i] < '0') || (str[*i] > '9'))		return 1;
		if(!IsAlphabet(str[*i+1]) && !IsNumber(str[*i+1]) && str[*i+1]!='(' && str[*i+1]!=')') return 1;
		//Add(Head, TRUE, str[*i]);
	}

	//Right part
	*i=OurPlace;
	//if(NotSpaceRight(str, i, len, &Hooks))		return 1;
	/*if((str[*i] < '(') || (str[*i] > '(' && str[*i] < '-') || (str[*i] > '-' && str[*i] < '0')
		|| (str[*i] > '9' && str[*i] < 'c') || (str[*i] > 'c' && str[*i] < 'l') || (str[*i] > 'l' && str[*i] < 's'))		return
1;
	*/
	return 0;
}

double ReadInt(char *str, int *i, int len)
{
//Add digitals to array"Reverse Poland Notation"

	char *c=(char*)malloc(1);
	int k;
	double result;

	for(k=0; ((str[*i]>47) && (str[*i]<58) || str[*i]=='.') && *i<len; k++, *i+=1)
	{
		c[k]=str[*i];
		c=(char*)realloc(c, k+2);
	}
	*i-=1;
	c[k]=0;
```

```
	if(k>10 && c[*i-k+1]=='-'|| k>9 && c[*i-k+1]!='-')
	{
		printf("\nError");
		getchar();
		exit(0);
	}
	result=atof(c);
	free(c);
	return result;
}

int IsZnak(char str)
{
	if(str=='*' || str=='+' || str=='-' || str=='/' || str=='^')return 1;
	return 0;
}

int IsNumber(char str)
{
	if(str >= '0' && str <= '9')	return 1;
	return 0;
}

int RepairCos(char *str, int *i, Priority **Dump, int write, int *Hooks)
{
	if(str[*i+1]=='o' && str[*i+2]=='s' && str[*i+3]=='(')
	{
		*i+=3;
		if(write)
			AddToDump(Dump, 'c', *Hooks);
		return 1;
	}
	return 0;
}

int RepairLog(char *str, int *i, Priority **Dump, int write, int *Hooks)
{
	if(str[*i+1]=='o' && str[*i+2]=='g' && str[*i+3]=='(')
	{
		*i+=3;
		if(write)
			AddToDump(Dump, 'l', *Hooks);
		return 4;
	}
	return 0;
```

```
}

int RepairSinSqrt(char *str, int *i, Priority **Dump, int write, int *Hooks)
{
	if(str[*i+1]=='i' && str[*i+2]=='n' && str[*i+3]=='(')
	{
		*i+=3;
		if(write)
			AddToDump(Dump, 's', *Hooks);
		return 2;
	}

	if(str[*i+1]=='q' && str[*i+2]=='r' && str[*i+3]=='t' && str[*i+4]=='(')
	{
		*i+=4;
		if(write)
			AddToDump(Dump, 'q', *Hooks);
		return 3;
	}

	return 0;
}

void Calculate(char *str, Priority **Dump, Priority *Nachalo, int *i, int len, Element **Head, int *Hooks, int fl)
{
	char c=str[*i];
	int iSysElement=*i;
	int iHooks=0;
	int iRepairZnak=0;
	int iFunc=0;
	int IsOpenHook=0;

	if(str[*i]=='-' && (IsZnak(str[*i-1]) || str[*i-1]=='(' || *i==0))//Repair on '-'
	{
		*i+=1;
		Add(Head, FALSE, -1*ReadInt(str, i, len));
		//if(iSysElement && !NotSpaceLeft(str, &iSysElement, 1) && IsZnak(str[iSysElement]))
	iSysElement=0;
		//if(iSysElement && str[iSysElement]=='(')		iSysElement=0;

	}else{
//		if(!IsZnak(c)){\
			*i+=1;\
			c=str[*i];\
		}
```

```
iFunc=0;
*i+=1;

//if(IsZnak(str[*i])) *i+=1;

switch (str[*i])
{
case 'c':	{
		AddToDump(Dump, c, *Hooks);
		iFunc=RepairCos(str, i, Dump, TRUE, Hooks);
		return;
				}
case 's':	{
		AddToDump(Dump, c, *Hooks);
		iFunc=RepairSinSqrt(str, i, Dump, TRUE, Hooks);
		return;
				}
case 'l':	{
		AddToDump(Dump, c, *Hooks);
		iFunc=RepairLog(str, i, Dump, TRUE, Hooks);
		return;
				}
case '(':	{
		AddToDump(Dump, c, *Hooks);
		Skobki(str, i, len, Hooks, Dump, Nachalo, Head);
		return;}
default:
		{
			if(IsNumber(str[*i]))
				Add(Head, FALSE, ReadInt(str, i, len));
			if(Dump[0]->cSymbol=='*' || Dump[0]->cSymbol=='/' || Dump[0]->cSymbol=='^')
				Add(Head, TRUE, ReadFromDump(Dump));
		}
}

if(c!=0)
		AddToDump(Dump, c, *Hooks);

if(fl==1 && (str[*i+1]!='+' || str[*i+1]!='-') && (c=='+' || c=='-'))
{
		*i+=1;
		Calculate(str, Dump, Nachalo, i, len, Head, Hooks, 1);
		return;
}
if(fl==0 && (str[*i+1]!='+' || str[*i+1]!='-') && (c=='+' || c=='-'))
```

```
		{
			*i+=1;
			Calculate(str, Dump, Nachalo, i, len, Head, Hooks, 1);
			return;
		}
	}
	Add(Head, TRUE, ReadFromDump(Dump));
}

char Repair(char *str, int len, Element *Head)
{
	int i;
	int Hooks=0;
	int ended=0;

	Priority *Dump=(Priority*)malloc(sizeof(Priority));
	Priority *Nachalo=Dump;
	Dump->past=NULL;
	Dump->cSymbol=0;
	Dump->next=NULL;

	for(i=0; str[i]!=0; i++)
	{
		switch (str[i])
		{
		case '(':	ended=Skobki(str, &i, len, &Hooks, &Dump, Nachalo, &Head);		break;
		case ')':
			ended=Skobki(str, &i, len, &Hooks, &Dump, Nachalo, &Head);		break;
		case '+':	{ended=Znaki(str, &i, len, FALSE, &Head);
			Calculate(str, &Dump, Nachalo, &i, len, &Head, &Hooks,0);		break;}
		case '-':	{ended=Znaki(str, &i, len, TRUE, &Head);
			Calculate(str, &Dump, Nachalo, &i, len, &Head, &Hooks,0);		break;}
		case '*':	{ended=Znaki(str, &i, len, FALSE, &Head);
			Calculate(str, &Dump, Nachalo, &i, len, &Head, &Hooks,0);		break;}
		case '/':	{ended=Znaki(str, &i, len, FALSE, &Head);
			Calculate(str, &Dump, Nachalo, &i, len, &Head, &Hooks,0);		break;}
		case '^':	{ended=Znaki(str, &i, len, FALSE, &Head);
			Calculate(str, &Dump, Nachalo, &i, len, &Head, &Hooks,0);		break;}
		case 'c':	RepairCos(str, &i, &Dump, TRUE, &Hooks);  break;
		case 's':	RepairSinSqrt(str, &i, &Dump, TRUE, &Hooks);  break;
		case 'l':	RepairLog(str, &i, &Dump, TRUE, &Hooks);  break;
		default:
			{
				if((str[i]>47) && (str[i]<58))
				{
```

```
                                        Add(&Head, FALSE, ReadInt(str, &i, len));
                                }
                        }
                }

                if(ended){              DestrPriority(&Dump);       return 0;}
        }
        while(Dump->past!=NULL)
                Add(&Head, TRUE, ReadFromDump(&Dump));
        if(Hooks!=0){       DestrPriority(&Dump);       return 0;}
        DestrPriority(&Dump);
}

int Movement(Element **Head)
{
        Element *Next=*Head;
        *Head=Next->next;
        if(Head[0]==NULL)           return 0;
        return 1;
}

int IsAlphabet(char str)
{
        if(64<str && str<91 || 96<str && str<123)       return 1;
        return 0;
}

int NeedToDo(Contar **Result, Element **Had, char operation)
{
//Calculate result
        Contar *Head=*Result;
        switch (operation)
        {
        case '+':
                {           Head->past->value+=Head->value;
                            *Result=Head->past;
                            free(Head);
                }break;
        case '-':
                {           Head->past->value-=Head->value;
                            *Result=Head->past;
                            free(Head);
                }break;
        case '*':
                {           Head->past->value*=Head->value;
```

```
                        *Result=Head->past;
                        free(Head);
                }break;
        case '/':
                {       if(Head->value==0)      return 0;
                        Head->past->value/=Head->value;
                        *Result=Head->past;
                        free(Head);
                }break;
        case '^':
                {       Head->past->value=pow(Head->past->value, Head->value);
                        *Result=Head->past;
                        free(Head);
                }break;
        case 'c':       {
        //              Movement(Had);
        //              AddToStack(Result, Had[0]->value);
                        Head->value=cos(Head->value);}          break;
        case 's':       {
        //              Movement(Had);
        //              AddToStack(Result, Had[0]->value);
                        Head->value=sin(Head->value);}          break;
        case 'q':       {
        //              Movement(Had);
        //              AddToStack(Result, Had[0]->value);
                        Head->value=sqrt(Head->value);}         break;
        case 'l':       {
        //              Movement(Had);
        //              AddToStack(Result, Had[0]->value);
                        Head->value=log(Head->value);}          break;
        }
        return 1;
}

int ReadToReplace(Replace **Top, char *str)
{
        Replace *Next=*Top;
        while(Next->past!=NULL)
        {
                if(!strcmp(Next->from, str))
                {
                        *Top=Next;
                        return 1;
                }else   Next=Next->past;
        }
```

```
	return 0;
}

int ReadToReplFunk(ReplFunk **Top, char *name, int iCountVar)
{
	ReplFunk *Next=*Top;
	while(Next->past!=NULL)
	{
		if(!strcmp(Next->name, name) && Next->iCountVar==iCountVar)
		{
			*Top=Next;
			return 1;
		}else	Next=Next->past;
	}
	return 0;
}

void DestrReplace(Replace **Top)
{
	Replace *Next;
	while(Top[0]->past!=NULL)
	{
		Next=Top[0];
		Top[0]=Top[0]->past;
		free(Next->from);
		free(Next->to);
		free(Next);
	}
	free(Top[0]);

}

void DestrReplFunk(ReplFunk **Top)
{
	ReplFunk *Next;
	while(Top[0]->past!=NULL)
	{
		Next=Top[0];
		Top[0]=Top[0]->past;
		free(Next->name);
		free(Next->str);
		free(Next);
	}
	free(Top[0]);
```

```
}

void DestrPriority(Priority **Top)
{
	Priority *Next;
	while(Top[0]->past!=NULL)
	{
		Next=Top[0];
		Top[0]=Top[0]->past;
		free(Next);
	}
	free(Top[0]);

}

void DestrContar(Contar **Top)
{
	Contar *Next;
	while(Top[0]->past!=NULL)
	{
		Next=Top[0];
		Top[0]=Top[0]->past;
		free(Next);
	}
	free(Top[0]);

}

void DestrElement(Element **Top)
{
	Element *Next;
	while(Top[0]->next!=NULL)
	{
		Next=Top[0];
		Top[0]=Top[0]->next;
		free(Next);
	}
	free(Top[0]);

}
```