学号:031602523 姓名:刘宏岩　学院:数计学院　专业:计算机类 班级:实验班

《Linux 操作系统设计实践》

实验五:图形界面

实验环境: Ubuntu16.04

实验内容: 四则运算自动出题器

1. 相关 API 学习

opgtk_window_new(GTK_WINDOW_TOPLEVEL)

- 函数创建一个窗口并返回这个窗口的控件指针。
- 参数 GTK_WINDOW_TOPLEVEL 指明窗口的类型为最上层的主窗口，它最常 用。还可以取另一个值 GTK_WINDOW_POPUP 指明窗口的类型为弹出式的无边框 的窗口。

g_signal_connect()

- 使用这个宏为窗口或控件加回调函数。g_signal_connect 宏有 4 个参数，分别是: 连接的对象，就是要连接信号的控件的指针(**注意:必须是已创建完的控件的指针**)，需要用 G_OBJECT 宏来转换; 信号名称，就是要连接的信号名称，为字符串形式，用双引号引起来。不同的控件拥有的信号名称是不一样的;
- 回调函数，指信号发生时调用的函数，这里只用到函数名称，需要用 G_CALLBACK 宏来转换一下;传递给回调函数的参数，它的值类型应该为 gpointer。如果不是这一类型 可以强制转换，如果没有参数则为 NULL。这里只能传递一个参数，如果有多 个参数，可以先将多个参数定义为一个结构，再将此结构作为参数传递过来

gtk_window_set_title(window,const gchar* title)

- 设定窗口的标题:

gtk_window_set_default_size(window,int width,int height)

- 设定窗口的默认宽高

gtk_window_set_position(window,GtkWindowPosition

position);

- 设定窗口的位置:
- 其中 position 可以取如下值

GTK_WIN_POS_NONE 不固定
GTK_WIN_POS_CENTER 居中
GTK_WIN_POS_MOUSE 出现在鼠标位置 GTK_WIN_POS_CENTER_ALWAYS 窗口改变尺寸仍居中 GTK_WIN_POS_CENTER_ON_PARENT 居于父窗口的中部

gtk_container_add ()

- 功能是将另一控件加入到容器中来。
- 它的第一参数是 GtkContainer 型的指针，这就需要将窗口控件指针用宏 GTK_CONTAINER 转换一下，即 GTK_CONTAINER(window)。它的第二参数是要容纳的控件的指针，即 button。

gtk_widget_show_all()

- 原本每一个控件都要用函数 gtk_widget_show 来显示，而这个函数显示容器中所有控件。
- 它的参数是一个容器控件的指针，例如本次实验中用到的：gtk_widget_show_all(window);

2. 实验思路

编写一个用于小学生学习四则运算的小程序，编写一个函数用于随机生成算式，和一个计算函数用于计算随机算式的结果。根据用户输入的答案进行对比，统计正确题目和错误题目的数目。

确认按钮：用于切换下一题，并判断结果对错。

清空按钮：清空计数器，还原程序。

3. 本次实验代码

```
#include <gtk/gtk.h>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string.h>
#include <stdlib.h>
#include <string>
#include <vector>
#include <stdlib.h>
#include <stdio.h>
#include <sstream>
#include <cstring>
#include <unordered_map>
#include <math.h>
#define random() (rand()%100000)

using namespace std;

string combine(string str1, string str2, char k)
```

```
{
        string combination;
        combination = str1 + k + str2;
        return combination;
}
string int_string(int number)
{
        int temp = abs(number);
        std::stringstream ss;
        std::string str;
        ss<<temp;
        ss>>str;
        return str;
}
int randomNunber()
{
        int a = random() % 10 + 1;
        return a;
}
char randomOperation()
{
        int j;
        char symbol[1];
        j = random() % 3;
        switch (j) {
        case 0:
                symbol[0] = '+';
                break;
        case 1:
                symbol[0] = '-';
                break;
        case 2:
                symbol[0] = '*';
                break;
        }
        return symbol[0];
}

string generateExpression()
{
        int num1, num2, change, count;
        char symbol;
        string str_num1, str_num2, Equation, t;
        num1 = randomNunber();
        num2 = randomNunber();
        count = random() % 6 + 2;
        symbol = randomOperation();
        str_num1 = int_string(num1);
        str_num2 = int_string(num2);
        Equation = combine(str_num1, str_num2, symbol);         //随机数与随机的符号结合
        for (count; count>2; count--) {
                symbol = randomOperation();
                str_num1 = Equation;
                change = random() % 2;
                if (change == 0) {
                    str_num1 = '(' + str_num1 + ')';
                }
                num2 = random() % 10 + 1;
                str_num2 = int_string(num2);
                change = random() % 2;
                if (change == 0) {
                    t = str_num1;
                    str_num1 = str_num2;
                    str_num2 = t;
                }
                Equation = combine(str_num1, str_num2, symbol);
        }
        //                  cout << Equation << "=" << endl;
        //string Equation2 = Equation + '=';
        //formula = Equation2;
        return Equation;
}
char priority(char pre, char post)
{
   if (pre == '+')
   {
      if (post == '+') return '>';
      else if (post == '-') return '>';
      else if (post == '*') return '<';
      else if (post == '/') return '<';
      else if (post == '(') return '<';
      else if (post == ')') return '>';
   }
   else if (pre == '-')
   {
      if (post == '-') return '>';
      else if (post == '+') return '>';
      else if (post == '*') return '<';
      else if (post == '/') return '<';
      else if (post == '(') return '<';
      else if (post == ')') return '>';
   }
   else if (pre == '*')
   {
      if (post == '*') return '>';
      else if (post == '/') return '>';
      else if (post == '+') return '>';
      else if (post == '-') return '>';
      else if (post == '(') return '<';
      else if (post == ')') return '>';
   }
   else if (pre == '/')
   {
      if (post == '/') return '>';
      else if (post == '*') return '>';
      else if (post == '+') return '>';
      else if (post == '-') return '>';
      else if (post == '(') return '<';
      else if (post == ')') return '>';
   }
```

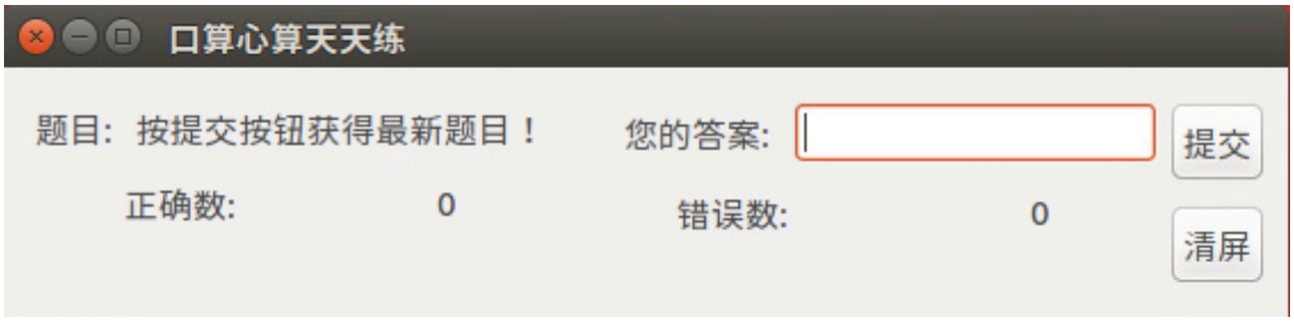
```
    else if (pre == '(')
    {
        if (post == '*') return '<';
        else if (post == '/') return '<';
        else if (post == '+') return '<';
        else if (post == '-') return '<';
        else if (post == '(') return '<';
        else if (post == ')') return '=';
    }
}

int caculate(int Operand1, int Operand2, char Operator) ;
int calculateResult(string str) {                    //用于计算计算生成算式的值
    vector< int > Operands;                          //运算数栈
    vector< char > Operators;                          //运算符栈
    int OperandTemp = 0;
    char LastOperator = 0;                           //记录上一次所遇到的符号
    for (int i = 0; i < str.size(); i++) {           //此循环用于去括号
        char ch = str[i];
        if ('0' <= ch && ch <= '9') {
            OperandTemp = OperandTemp * 10 + ch - '0';
        }
        else if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '(' || ch == ')') {
            if (ch != '(' && LastOperator != ')') {        //结合本次和上次所遇见的符号来判断是否
需要将当前存储的运算数压入栈
                Operands.push_back(OperandTemp);
                OperandTemp = 0;
            }
            char Opt2 = ch;
            for (; Operators.size() > 0;) {
                char Opt1 = Operators.back();
                char CompareRet = priority(Opt1,Opt2);   //用当前符号与栈顶符号来对算式简化
                if (CompareRet == '>') {                 //当前的符号的优先级小于栈顶符号时就可
以将栈顶符号计算掉并将结果压入栈
                    int Operand2 = Operands.back();
                    Operands.pop_back();
                    int Operand1 = Operands.back();
                    Operands.pop_back();
                    Operators.pop_back();
                    int Ret = caculate(Operand1, Operand2, Opt1);
                    Operands.push_back(Ret);
                }
                else if (CompareRet == '<') {              //当前的符号优先级大于栈顶符号不能进
行运算所以跳出循环来存储当前符号
                    break;
                }
                else if (CompareRet == '=') {              //这个是“（”,“）”结合的情况 所以移除
“（”并退出循环
                    Operators.pop_back();
                    break;
                }
            }
            if (Opt2 != ')') {
                Operators.push_back(Opt2);
            }
            LastOperator = Opt2;
        }
    }
    if (LastOperator != ')') {                              //接下来就是计算一个不含括号的算式了
        Operands.push_back(OperandTemp);
    }
    for (; Operators.size() > 0;) {
        int Operand2 = Operands.back();
        Operands.pop_back();
        int Operand1 = Operands.back();
        Operands.pop_back();
        char Opt = Operators.back();
        Operators.pop_back();
        int Ret = caculate(Operand1, Operand2, Opt);
        Operands.push_back(Ret);
    }
    return Operands.back();                                  //返回结果
}
int caculate(int Operand1, int Operand2, char Operator) {       //计算函数
    int result = 0;
    if (Operator == '+') {
        result = Operand1 + Operand2;
    }
    if (Operator == '-') {
        result = Operand1 - Operand2;
    }
    if (Operator == '*') {
        result = Operand1*Operand2;
    }
    if (Operator == '/') {
        result = Operand1 / Operand2;
    }
    return result;

}

GtkWidget *lb_problem;
GtkWidget *lb_correct;
GtkWidget *lb_wrong;
GtkWidget *lb_ans;

int correctNum = 0;
int wrongNum = 0;
string lastProb = "";

void refreshProblem()
{
}

void onBtnSubmit(GtkWidget *widget, gpointer data)
{
	if(lastProb == "")
	{
		lastProb = generateExpression();
		cout << lastProb << endl;
		gtk_label_set_text((GtkLabel*)lb_problem,lastProb.c_str());

		//ans = calculateResult(s);
		gtk_entry_set_text((GtkEntry *)lb_ans,"");
	}
	else
	{
		const gchar *myans = gtk_entry_get_text((GtkEntry *)lb_ans);
		if(atoi(myans) == calculateResult(lastProb))
		{

	gtk_label_set_text((GtkLabel*)lb_correct,int_string(++correctNum).c_str());
			cout << "Yes" << endl;
		}
		else
		{

	gtk_label_set_text((GtkLabel*)lb_wrong,int_string(++wrongNum).c_str());
			cout << "No" << endl;
		}
		gtk_entry_set_text((GtkEntry *)lb_ans,"");
		lastProb = generateExpression();
		cout << lastProb << endl;
		gtk_label_set_text((GtkLabel*)lb_problem,lastProb.c_str());
		const char *myAns = gtk_entry_get_text((GtkEntry *)lb_ans);
	}

}

void onBtnClear(GtkWidget *widget, gpointer data)
{
	lastProb = "";
	gtk_entry_set_text((GtkEntry *)lb_ans,"");
	gtk_label_set_text((GtkLabel*)lb_problem,"按提交按钮获得最新题目！");
	correctNum = 0;
	wrongNum = 0;
	gtk_label_set_text((GtkLabel*)lb_correct,"0");
	gtk_label_set_text((GtkLabel*)lb_wrong,"0");
}

void UI(int argc, char *argv[])
{
	gtk_init(&argc,&argv);
	GtkWidget *window=gtk_window_new(GTK_WINDOW_TOPLEVEL);
	gtk_window_set_default_size(GTK_WINDOW(window),200,50);
	g_signal_connect(G_OBJECT(window),"delete_event",G_CALLBACK(gtk_main_quit
),NULL);
```

```
gtk_window_set_title(GTK_WINDOW(window),"口算心算天天练");
gtk_window_set_position(GTK_WINDOW(window),GTK_WIN_POS_CENTER);
gtk_container_set_border_width(GTK_CONTAINER(window),10);

//----------------------------------------------------
GtkWidget *hb = gtk_hbox_new(0,0);
gtk_container_add(GTK_CONTAINER(window),hb);

//----------------------------------------------------
GtkWidget *vbox = gtk_vbox_new(0,0);
gtk_container_add(GTK_CONTAINER(hb),vbox);
//----------------------------------------------------
GtkWidget *hbox1 = gtk_hbox_new(0,0);
gtk_box_pack_start(GTK_BOX(vbox),hbox1,FALSE,FALSE,5);

GtkWidget *lb4 = gtk_label_new("题目:");
gtk_box_pack_start(GTK_BOX(hbox1),lb4,1,0,5);

lb_problem = gtk_label_new("按提交按钮获得最新题目！");
gtk_box_pack_start(GTK_BOX(hbox1),lb_problem,1,0,5);

GtkWidget *hbox11 = gtk_hbox_new(0,0);
gtk_box_pack_start(GTK_BOX(vbox),hbox11,FALSE,FALSE,5);

GtkWidget *lb2 = gtk_label_new("正确数:");
gtk_box_pack_start(GTK_BOX(hbox11),lb2,1,0,5);

lb_correct = gtk_label_new("0");
gtk_box_pack_start(GTK_BOX(hbox11),lb_correct,1,0,5);
//----------------------------------------------------
GtkWidget *vb3 = gtk_vbox_new(0,0);
gtk_container_add(GTK_CONTAINER(hb),vb3);

GtkWidget *hb4 = gtk_hbox_new(0,0);
gtk_box_pack_start(GTK_BOX(vb3),hb4,FALSE,FALSE,5);

GtkWidget *lb = gtk_label_new("\t 您的答案:");
gtk_box_pack_start(GTK_BOX(hb4),lb,1,0,5);

lb_ans = gtk_entry_new();
gtk_box_pack_start(GTK_BOX(hb4),lb_ans,1,0,5);

GtkWidget *hbox2 = gtk_hbox_new(0,0);
gtk_box_pack_start(GTK_BOX(vb3),hbox2,FALSE,FALSE,5);

GtkWidget *lb3 = gtk_label_new("错误数:");
gtk_box_pack_start(GTK_BOX(hbox2),lb3,1,0,5);

lb_wrong = gtk_label_new("0");
gtk_box_pack_start(GTK_BOX(hbox2),lb_wrong,1,0,5);

//----------------------------------------------------
```

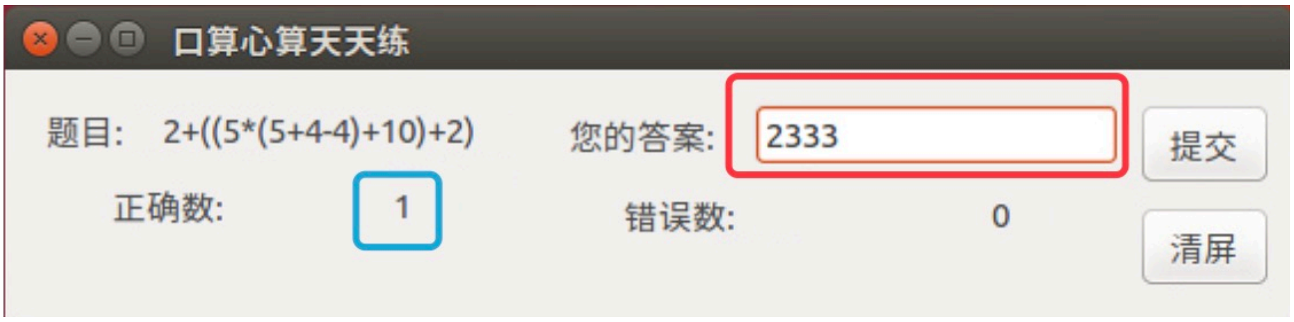
```
        GtkWidget *vb = gtk_vbox_new(0,0);
        gtk_container_add(GTK_CONTAINER(hb),vb);

        GtkWidget *btn_submit = gtk_button_new_with_label("提交");
        gtk_box_pack_start(GTK_BOX(vb),btn_submit,1,0,5);
        g_signal_connect(G_OBJECT(btn_submit),"clicked",G_CALLBACK(onBtnSubmit),N
ULL);

        GtkWidget *btn_clear = gtk_button_new_with_label("清屏");
        gtk_box_pack_start(GTK_BOX(vb),btn_clear,1,0,5);
        g_signal_connect(G_OBJECT(btn_clear),"clicked",G_CALLBACK(onBtnClear),NULL
);


        //--------------------------------------------------------
        gtk_widget_show_all(window);//显示所有窗体
        gtk_main();
}

int main(int argc, char *argv[])
{
        srand(time(0));

        UI(argc,argv);

        double ans = 0;
        string s = "";
        s = generateExpression();
        cout << "所产生的算式为" << s << endl;
        ans = calculateResult(s);
        cout << "结果为" << ans << endl;
        return 0;
}
```

4. **实验截图**

- 程序初始化



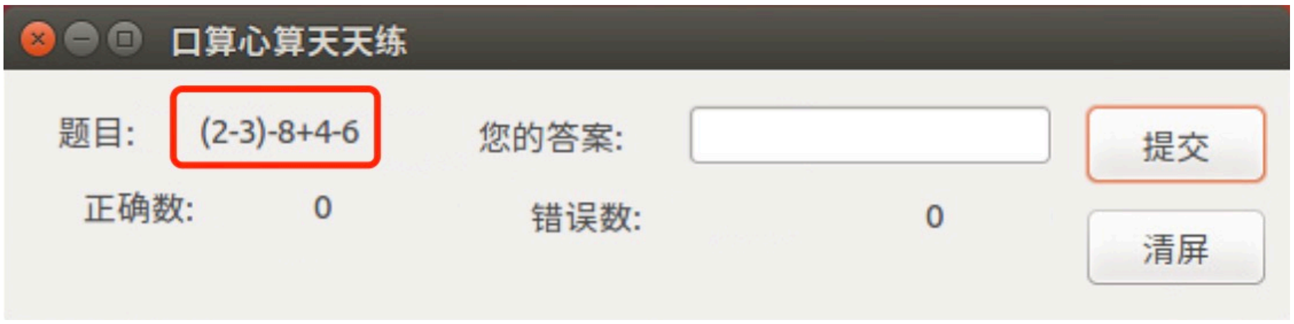
- 出题



题目: 2+((5*(5+4-4)+10)+2)　　您的答案: 2333

正确数: 1　　错误数: 0

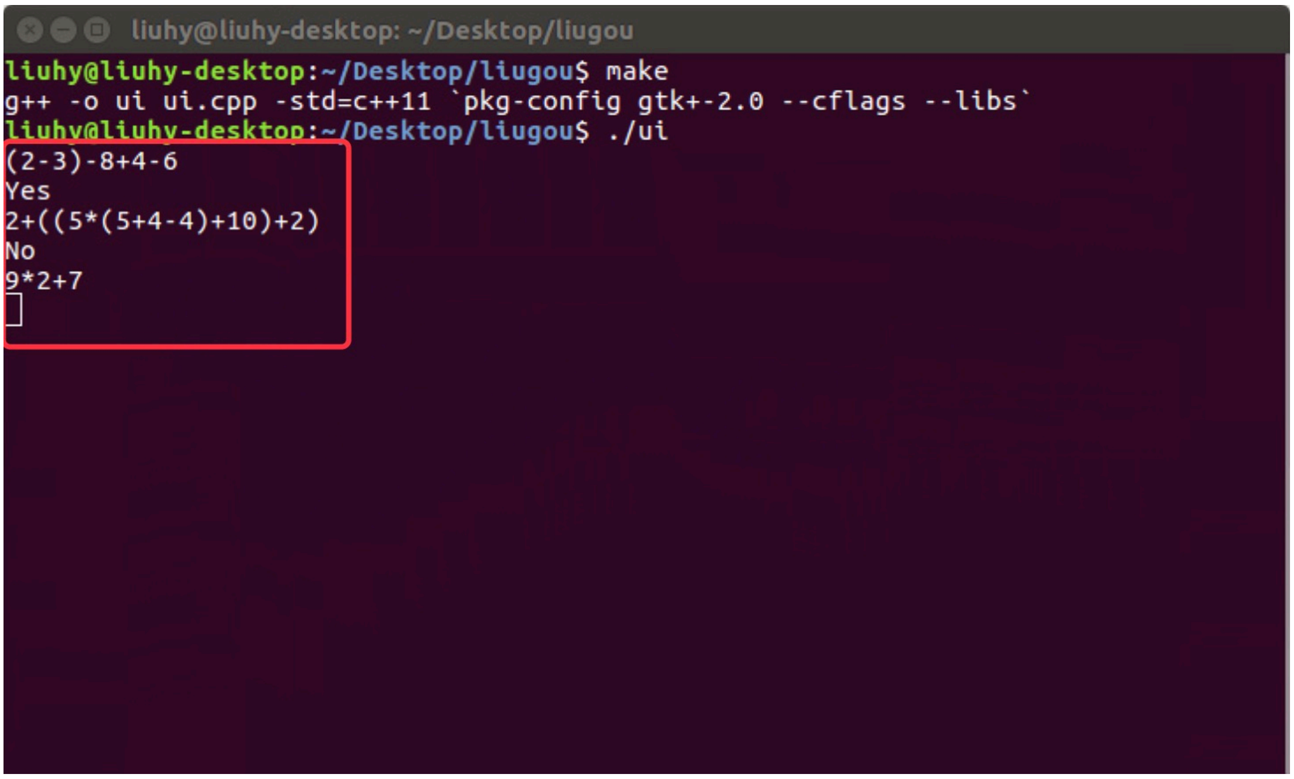
● 输入正确结果后提交，正确数加一



题目: (2-3)-8+4-6　　您的答案:

正确数: 0　　错误数: 0

● 在终端中统计题目



```
liuhy@liuhy-desktop: ~/Desktop/liugou
liuhy@liuhy-desktop:~/Desktop/liugou$ make
g++ -o ui ui.cpp -std=c++11 `pkg-config gtk+-2.0 --cflags --libs`
liuhy@liuhy-desktop:~/Desktop/liugou$ ./ui
(2-3)-8+4-6
Yes
2+((5*(5+4-4)+10)+2)
No
9*2+7
```