Document of the unnamed gui-lib

关于图形库的一份说明

目录

关于这个无名库

本图形库是一个基于 graphics 库、从面向对象的概念出发的图形库：图形库的实现将所需要绘制的 gui 单元视为一个对象（object），每个对象有自己的结构属性、绘图方式、对于外界信号的响应方式，以及附加信息；对象可以向系统注册或者删除所需要接受的信号模式；对象可以组成对象组，对象组也可以作为某个对象的成员；对象和对象之间可以通过回调函数，也可以通过信号与槽的方式进行多对多的通信。

实现这个库的起因是因为…虽然对于《c 语言的科学和艺术》书中提供的图形库的 api 以及一些绘图实现方式做了一定程度上的更新，但是更新后的图形库依然保持简单的过程式编程范式、使用全局的回调函数进行消息处理的模式，仅仅提供了对于底层 api 简单的一层封装，很大程度上不利于复杂的 gui 软件的编写。因此，在此基础上，我想开发一个简易的、包含面向对象思想的图形库，提供更高程度的抽象，使代码结构更加清晰，也简化开发流程。当然，实际的实现还是非常粗糙和不完善的。（不过也是挺好玩的事情）

其中一些想法来源于 java 的 swing 组件，另一些大概来自于 windows 的回调函数、消息队列机制，以及 qt 库等等 gui 实现，还有自己用过的 pygame 之类的 2d 游戏引擎之类的

小玩意。

PART1：基本概念和数据结构：

对象：

对象是整个图形库的核心概念，其数据结构定义如下：

```
//the struct of an obj
struct objo{
    paintMethod paint; //the additional paint method
    updateMethod update; //the pointer for update function
    void* structure; //the structure of the obj, like the rectangle struct
    operation op; //the operation pointer
    void* source; //the additional source for some data,like pictures,string, you can
define your own data struct
    List node; //the list node for display
    int flag; //flag 后四位是指明 object type ,中间5位是event type
};
//object type pointer
typedef struct objo Object;
typedef Object* OBJECT;
```

在 c 语言中，我使用了结构体来定义了一个对象包含的数据。(如果在 oop 中，大概对应一个抽象类之类的玩意)

结构体中包含三个函数指针，分别对应该对象的三种方法：paintMethod 指对象的绘图方法； updateMethod 指对象可随时间改变的更新方法；operation 为接受相关消息，如鼠标键盘等的相应方法。三种方法都作为回调函数被库自动调用。

```
typedef void (*paintMethod)(Object* obj);//paint
typedef void (*updateMethod)(Object* obj);
typedef int (*operation)(Object* obj,int x,int y,int button,int Event,int eventKind);
```

结构体同样包含两个空指针保存相关数据块，structure 指针指向的数据块包含对象的大小、位置、形状等相关信息，有预定义好的椭圆、多边形、长方形三种类型，同样也可以保存自定义相关结构体；source 保存对象的一些附加信息，如字符串、图片元素、声音元素等等，自定义的一些属性信息也推荐存放在这个指针指向的数据块中。

同时，我们也提供了相关的宏定义以方便的进行空指针的类型转换。

```
#define STRUCTURE(TYPE,obj) ((TYPE)(obj->structure))
#define SOURCE(TYPE,obj) ((TYPE)(obj->source))
```

Flag 其中按二进制位保存一些相关信息，如该对象的类型、懒惰删除的标签等等。具体可查阅相关宏定义。

同样可以使用宏定义获取 flag 中的信息，如获取对象的结构类型：

```
#define TYPEFLAG(obj) (obj->flag&ALL_TYPE)
```

List node 保存指向全局对象表中保存该对象的链表节点。(冗余设计)

另外，在本库的实现中，所有的宏定义和指针类型一律全部大写，结构体部分小写，后接字母 S；函数采用驼峰命名法。

对象的默认结构

我们定义的对象有三种预定义结构和一种自定义结构，结构信息保存在 structure 指针指向的数据块中。其中主要有三种参数:位置信息,对于矩形和椭圆来说是左上角点的坐标、宽度和高度，对多边形而言是顶点；color，绘制颜色；brush style 笔刷类型。

```
//the rectangle structrure
typedef struct {
    int posX;
    int posY;
    int height;
    int width;
    Color color;
    ACL_Brush_Style style;
} rectS;

//the ellipseS structrure: nearly the same as the rectangle structrure
typedef struct {
    int posX;
    int posY;
    int height;
    int width;
    Color color;
    ACL_Brush_Style style;
} ellipseS;

//the polygon structure
typedef struct {
    POINT *points;
    int pointCount;
    Color color;
    ACL_Brush_Style style;
} polygonS;

//the suructure for other obj type
typedef struct {
    void* data;
    Color color;
    ACL_Brush_Style style;
```

```
} otherObj;
```

结构体名称小写，大写的为结构体指针；

```
typedef rectS* RECTANGLE;
typedef ellipseS* ELLIPSE;
typedef polygonS* POLYGON;
typedef otherObj* OTHERSHAP;
```

应注意的是，椭圆和矩形的结构体是基本相同的，因此指针和函数某种程度上可以通用，虽然不推荐这样。

对于三种预定义结构，它们的绘图方法由系统自动完成，可以不必指定；自定义结构必须提供绘制方法。另外，如果对预定义结构提供了绘图方法，绘图方法将在系统绘制完基础结构图像之后调用。

初始化函数

```
void setup();
```

您在创建菜单或者创建程序开始时候就出现的对象必须在这个函数中完成。如果什么都不需要初始化，最好也把它写上去。

绘图方法:

本图形库系统的绘图模式是刷新模式，即每隔一定时间，将屏幕清空并重新绘制所有图形。初始设置是 30ms 一次，也可以通过 api 来重新设定。如果您要绘制相应图形，可以使用预定义好的结构对象，也可以创建一个绘图对象进行绘图。详见 api 函数。

对象的消息处理回调函数:

如果要让对象处理系统传来的键盘、鼠标等消息，需要实现如下回调函数，并附加到变量中:

```
typedef int (*operation)(Object* obj,int x,int y,int button,int Event,int eventKind);
```

operation 函数的返回值可以作为是否捕获该消息的判断标准；若返回 1，则消息不会被传递给消息传递队列中下一个对象，也不会交由全局函数处理。该函数需要由使用者自己实现，各参数的含义分别是:

当鼠标移动或点击时，x,y 为点击坐标，button 为所按鼠标键，event 的选项包含在 ACL_Mouse_Event 中，eventkind 为以下三种之一 MOUSEDOUBLECLICK_EVE，MOUSEDOWN_EVE: MOUSEUP_EVE;

当键盘按键时，button 对应所按下的 key，event 为 KEY_DOWN 或 KEY_UP，eventkind 为 KEY_EVE，其他参数无意义;

当接收字符消息时，button 对应 char，eventkind 为 CHAR_EVE，其他参数无意义。

消息处理回调的具体实现中，我们采用链表进行组织: 每种类型的消息都有一个独立的链表，通过遍历该链表给表中每一个已注册该类型消息处理回调函数的对象发送消息。对象注册对于某个消息的响应（调用 setCall 函数），就是在链表中插入节点；取消响应（调用

releaseCall 函数)，就是移除链表中的节点。为了保证时间复杂度，我们使用了懒惰删除的方法进行节点移除。

对象组

本图形库支持创建相关的对象组，并对对象组中的对象执行随机存取、遍历、删除等操作。对象组也可以作为对象的 source 附加信息。

数据结构定义如下

```
//the set of objects
typedef struct {
    OBJECT *objs;
    int num;
    int maxSize;
} objectSet;
//pointer defines
typedef objectSet* OBJECTS;
```

对象组的创建、删除、存取等请参见 api 部分。

信号与槽机制

信号槽是设计模式中观察者模式的一种实现，或者说是一种升华。一个信号就是一个能够被观察的事件，或者至少是事件已经发生的一种通知；一个槽就是一个观察者，通常就是在被观察的对象发生改变的时候——也可以说是信号发出的时候——被调用的函数；你可以将信号和槽连接起来，形成一种观察者-被观察者的关系；当事件或者状态发生改变的时候，信号就会被发出；同时，信号发出者有义务调用所有注册的对这个事件（信号）感兴趣的函数（槽）。

信号和槽是多对多的关系。一个信号可以连接多个槽，而一个槽也可以监听多个信号。信号也可以有附加信息。

这里提供了一种简单的信号/槽的实现方式：您可以创建一个槽，把它和特定的对象和响应函数进行关联；也可以创建一个信号，和相应的对象进行关联。通过槽和信号的连接，当信号从信号源被发出时，相应的槽就会被自动回调。具体使用方式详见 api。

PART2：Api 函数

创建和删除对象：

创建和删除对象是本库实现的基础。任何调用都应该尽量通过对象表达。

```
OBJECT createObj(int objType, paintMethod paint, void *structure, operation op, int callType);
```

该函数用于创建一个对象，返回指向对象的指针。

参数：

int ObjType：对象的类型：可以是以下四种对象之一：前三种内置类型系统可以自动绘制而无需指定paint参数。

RECT_TYPE 矩形类型

ELLI_TYPE 椭圆

POLY_TYPE 多边形

OTHER_TYPE 其他

paintMethod paint：函数指针，指向该对象的绘制方法。

void *structure：该对象的结构

operation op：该对象的消息处理函数指针

int callType：该对象注册的接受消息类型：多个参数可以使用 | 进行连接

CHAR_EVE 响应字符消息

MOUSEMOVE_EVE 响应鼠标移动消息

MOUSECLICK_EVE 响应点击消息，如果是内置类型的话，只有鼠标在形状内部点击才会触发

KEY_EVE 响应键盘消息

示例：`RECTANGLE r1 = createRect(10, 10, 30, 80, WHITE);`

`OBJECT obj1 = createObj(RECT_TYPE, NULL, r1, NULL, 0);`

创建一个矩形结构，然后用此矩形结构创建一个对象。

`void deleteObj(OBJECT obj);`

该函数用于删除一个对象，无返回值。应注意删除后的对象指针已无意义。删除对象的时候不采用懒惰删除方式，需要遍历链表找到需要删除的对象，因此时间开销较高；如非必要，不建议过多调用删除和创建对象，可设置对象为不显示或不响应相关消息。

参数：对象指针。

创建内置对象的结构

如果想使用内置对象结构类型，在创建对象之前需要先创建相关结构。

`RECTANGLE createRect(int x,int y,int height,int width, Color c);`

该函数创建一个矩形对象结构，返回指向矩形对象结构的指针。

参数：

int x,int y,int height,int width 左上角坐标和宽度、高度；

Color c；颜色

`ELLIPSE createEllipse(int x, int y, int height, int width, Color c);`

该函数创建一个椭圆对象结构，返回指向椭圆对象结构的指针。

参数：

int x,int y,int height,int width 左上角坐标和宽度、高度；

Color c；颜色

`ELLIPSE createRound(int x, int y, int r, Color c);`

该函数创建一个圆形对象结构，返回指向椭圆对象结构的指针。

参数:

int x,int y,int r 左上角坐标和半径;

Color c; 颜色

```
POLYGON createPolygon(POINT *points, Color c, int pointNum);
```

该函数创建一个多边形对象结构，返回指向椭圆对象结构的指针。

参数:

POINT *points ,点的坐标数组指针

Color c; 颜色

int pointNum 点的数量

对象组相关函数

```
//functions for object set
OBJECTS createObjSet();
```

该函数创建一个对象组，返回指向对象组的指针。

```
void addSetObj(OBJECTS objset, OBJECT obj);
```

该函数将一个对象添加进对象组中。对象组本质是一个类似 vector 的变长数组，当添加的对象数量大于对象组容量时会自动扩容。

参数:

OBJECTS objset,对象组指针

OBJECT obj 对象指针

```
void deleteObjSet(OBJECTS objset);
```

该函数删除一个对象组。

参数为对象组指针。

```
OBJECT getObjInSet(OBJECTS objset, int i);//get the OBJ at pos i;
```

获取对象组中下标为 i 的对象，返回值为对象指针。当下标越界时返回 NULL。数组的当前容量可使用 objset->num 查询。

参数:

OBJECTS objset 对象组

int I 下标

```
void deleteObjInSet(OBJECTS objset,OBJECT obj);
```

删除对象组中的对象

参数:

OBJECTS objset 对象组

OBJECT obj 要删除的对象

给对象附加或改变属性信息:

这一类函数处理对象的附加信息，其中大部分是定义在对象结构体中的。

```
void addsource(OBJECT obj,void* source);
```

给对象添加附加信息

参数:

OBJECT obj 对象

void* source 附加信息的指针

```
void setUpdate(OBJECT obj,updateMethod update);
void cancelUpdate(OBJECT obj);
```

给对象添加或删除随时间更新的方法。

```
void addpaintMethod(OBJECT obj, paintMethod paint);
void cancelpaintMethod(OBJECT obj);
```

给对象添加或删除绘制方法。

```
void changeObjectOp(OBJECT obj, operation op);
```

改变对象的消息响应回调函数。

```
void setNotDisplay(OBJECT obj);
void setDisplay(OBJECT obj);
```

设置对象是否在屏幕上显示。

```
void setCalls(int callType, OBJECT obj);
void releaseCalls(int callType, OBJECT obj);
```

设置对象所接收的消息类型，其中 int callType 参数如下:

int callType: 该对象注册的接受消息类型: 多个参数可以使用 | 进行连接

CHAR_EVE 响应字符消息

MOUSEMOVE_EVE 响应鼠标移动消息

MOUSECLICK_EVE 响应点击消息，如果是内置类型的话，只有鼠标在形状内部点击才会触发

KEY_EVE 响应键盘消息

调用 releaseCalls 函数时，采用懒惰删除的方法。

时间函数

本实现不提供定时器接口，但有一个全局计时器: 从程序开始运行记时。可以通过在 update 方法中调用如下函数完成对象随时间变化:

```
double getTime();
```

获取全局时间信息，返回值为 double 类型，从程序开始运行记时，单位为 s。

```
void resetTime();
```
重新设置全局计时器从 0 开始。

```
void setFrequency(int ms);
```
设置显示刷新频率，单位为 ms。初始设置为 30ms，不建议太快，太快会影响性能。

注册默认回调函数

默认回调函数只有在所有相关对象的消息处理函数被调用后，并且没有对象捕获消息时生效。具体参数定义可参照 graphics 库。

```
void setDefaultKeyboardEvent(KeyboardEventCallback callback);
void setDefaultCharEvent(CharEventCallback callback);
void setDefaultMouseEvent(MouseEventCallback callback);
void setDefaultTimerEvent(TimerEventCallback callback);
```

判断两个对象是否重叠

如果您要编写小游戏的话，这个功能有很大程度上会用到。

```
bool isOverlap(OBJECT obj1, OBJECT obj2);
```
该函数暂时只对内置类型生效，返回一个 bool 值判断两个对象是否在屏幕坐标系中重叠。参数为两个对象的指针。你

封装好的组件：图片、文本框、绘图器

这个部分提供了一些已经封装好的组件函数，比如文本框，图片对象，绘图生成器等等。

```
OBJECT createImageObj(string name, int x, int y, int height, int width,operation op);
```
该函数创建一个矩形的图片对象，返回一个对象指针。

参数：

string name：文件名，如果不在同一个文件夹下可使用相对路径。

int x, int y, int height, int width,：图片放置的大小和坐标；

operation op；和图片对象相关的操作回调函数

```
OBJECT setTextBox(OBJECT obj, string name, int maxChar, bool editable);
```
该函数将一个矩形或椭圆形的对象设置为文本框，返回一个对象指针。设置成文本框后，对象原先的 operation 回调函数除了字符处理部分依旧生效，字符消息会被当做文本框的输入处理。

参数：

OBJECT obj：需要设置为文本框的对象
string name：文本框的内容
int maxChar,：文本缓冲区的大小
bool editable：文本框是否能被编辑

```
void changeTextBox(OBJECT textBox, string text, bool editable);
```

该函数改变文本框的内容或可编辑状态。

参数：

OBJECT textBox：文本框对象，需要先调用 setTextBox 将对象转变为文本框对象。
string text：文本框的内容
bool editable：文本框是否能被编辑

```
OBJECT createDrawers(paintMethod paint, void *structure);
```

该函数创建一个用来绘制图形的空白对象，返回一个对象指针。本库的实现要求将所有的绘图和处理部分封装成对象，该函数可以用来创建一个仅仅用于绘图的没有其他功能的对象。

参数：

paintMethod paint：需要实现的绘图函数
void *structure：和对象相关联的绘图所需信息

信号与槽

关于信号和槽的简单实现：

```
SLOT createSlot(OBJECT obj, void(*receiver)(OBJECT obj, void* message));
```

该函数创建一个槽，将接受信号的函数和相应对象进行绑定，并返回一个指向槽的指针。

参数：

OBJECT obj：绑定的对象
void(*receiver)(OBJECT obj,void* message)；接受消息的函数指针：带有两个参数，其一是该函数所绑定的对象，其二是一个空指针类型用以传递自定义消息；该函数由系统自动回调。

```
SIGNAL createSignal(OBJECT obj);
```

该函数创建一个信号，返回一个指向信号结构体的指针。参数为所绑定的对象。

```
void connectSS(SLOT slot,SIGNAL signal);
```

该函数将信号和槽进行绑定；一个信号可以和任意数量槽绑定，一个槽也可以和任意数量信号进行绑定。

参数分别为指向槽的指针和指向信号的指针。

```
void sentSignal(SIGNAL signal,void* message);
```

该函数由一个信号源发出信号。信号可以附带消息，可将消息的指针信息存在 message 变量中。系统会自动调用所有和信号绑定的槽的回调函数并传递消息。

```
void disconnectSS(SLOT slot, SIGNAL signal);
```
该函数将一个信号和一个槽解绑。

```
void deleteSlot(SLOT slot);
```
该函数删除一个槽。注意，在删除槽之前必须将所有和该槽绑定的信号解绑。

```
void deleteSignal(SIGNAL signal);
```
该函数删除一个信号。

PART3：测试和示例

为了保证这个库系统的正确运行，我对每个组件进行了单元测试，同时综合起来实现了一个小 cad 的 demo。测试样例可以在 sample 文件夹下找到。