

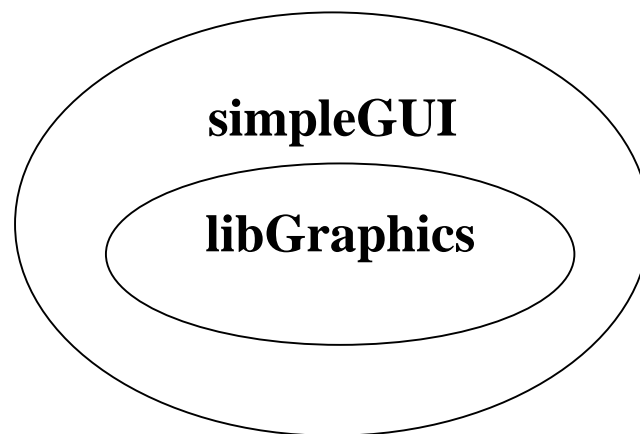


# 专题三

## 图形程序设计

# 专题三 图形程序设计

- 基本图形编程
- 交互图形编程
- **simpleGUI**介绍



# LibGraphics

- **boolean.h**

- 布尔变量 (bool: TRUE(1), FALSE(0))

- **exception.h/.c**

- 异常处理 (C++ try catch)

- **gcalloc.h**

- 支持垃圾回收的动态内存申请 (Java)

- **genlib.h/.c**

- 类型申明, 内存申请, 错误处理, repeat宏定义

- **simpio.h/.c**

- 简化数据读取

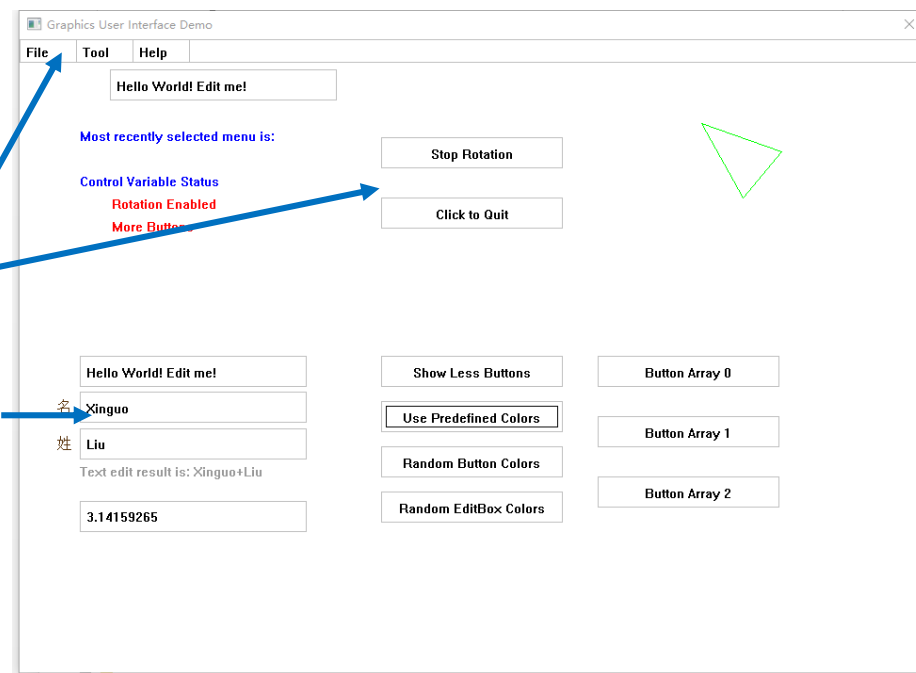
# LibGraphics

- random.h/.c (建议阅读)
  - 随机初始化, 随机整数, 随机浮点数
- strlib.h/.c (建议阅读)
  - 字符串处理
- linkedlist.h/.c (建议阅读)
  - 带头结点的链表
- graphics.h, extgraph.h, graphics.c (阅读.h)
  - 图形绘制与交互, 消息机制
  - 理解原理, 能够使用类库函数

# simpleGUI

## ■ imgui.h/.c (阅读.h)

- menuList – 菜单列表
- button – 按钮
- textbox – 编辑文本框



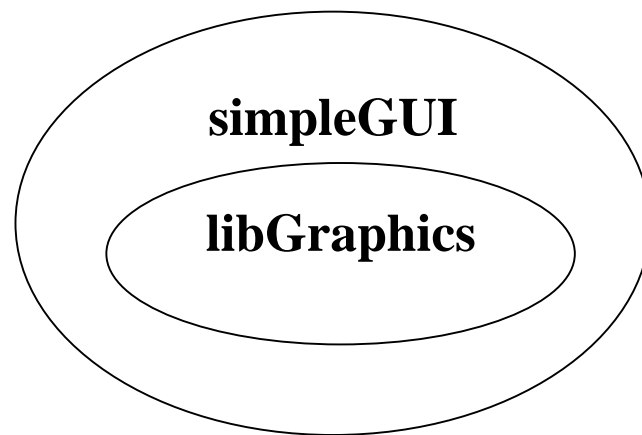
# 专题三 图形程序设计

## ■ 基本图形编程

- Win32程序 (WinMain – Main - InitGraphics)
- Win32工程创建 (Dev-C++ 和 VS2010+)
- 窗口相关函数
- 基本绘图函数 (画笔 - 线 - (椭)圆/弧 - 文本 - 清屏)
- 基本绘图属性 (颜色, 线宽, 区域填充, 字体)

## ■ 交互图形编程

## ■ simpleGUI介绍



# 第三方图形库

- C语言本身不提供图形绘制功能
  - 借助于第三方提供的图形库，可实现图形的绘制
- 图形库以C原码形式(.h/.c)，或者以二进制目标码形式(.h/.lib)提供
  - 在应用第三方图形库时，不需要了解其具体的实现，只需了解其基本功能和图形绘制流程
  - 直接调用相关图形库函数来实现具体的图形绘制
  - 头文件包含了相关图形库函数的原型
  - 图形库接口——头文件应当被包含到源文件中（工程文件实现程序文件模块连接）

# Windows API

- 本课程采用的第三方图形库是基于Windows系统的——基于Win32API
- 在Win32API中，第一个C函数是int **WinMain()**，而不是int main()，且要遵循Windows编程规范——这需要花很多时间去学习
- 为了方便初学者使用，在第三方图形库中，已实现了通用的int WinMain()基本功能，而应用程序所要做的相关初始化工作只需写在**void Main()**函数中即可



# WinMain函数

## ■ WinMain函数(类库)调用Main函数(自己实现)

```
int WINAPI WinMain (HINSTANCE hThisInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszArgument, int nFunsterStil)
{
    MSG messages; /* Here messages to the application are saved */

    Main();

    /* Run the message loop. It will run until GetMessage() returns 0 */
    while (GetMessage (&messages, NULL, 0, 0)) {
        /* Translate virtual-key messages into character messages */
        TranslateMessage(&messages);
        /* Send message to WindowProcedure */
        DispatchMessage(&messages);
    }
    FreeConsole();
    return messages.wParam;
}
```

graphics.c: 2050~2070

# Main函数

- 在Main()函数中，首先要调用InitGraphics()来初始化图形窗口，以便绘制图形

□ void InitGraphics(void)

```
void InitGraphics(void)
{
    if (!initialized) {
        initialized = TRUE;
        ProtectVariable(stateStack);
        ProtectVariable(windowTitle);
        ProtectVariable(textFont);
        InitColors();
        InitDisplay();
    }
    DisplayClear();
    InitGraphicsState();
}
```

graphics.c: 318~331

# Win32工程创建

## ■ Visual Studio 2010+

- libgraphics/docproject

## ■ Dev-C++ (每个项目新建一个目录)

- 新建项目选Empty Project, C Project

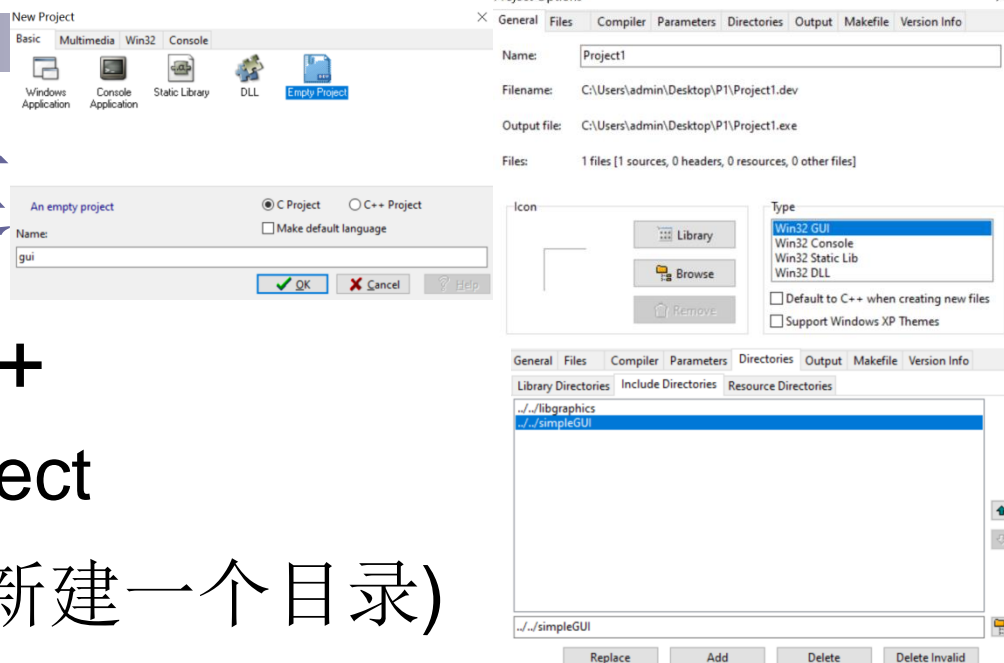
- Project->Project Options

- General中选择Win32 GUI

- Directories中Include Directories, 增加libgraphics和simpleGUI路径 (建议使用相对路径)

- Outputs中编译和程序位置也可以设置, 如../compiled或./output

- 增加libgraphics和simpleGUI的类库文件 (如果.h/.lib?)



# 窗口相关函数 - 1

- **void InitGraphics(void)**

- 初始化图形窗口，以便绘制图形

- **void InitConsole(void)**

- 打开控制窗口，方便程序调试
  - 使用scanf/printf进行quick & dirty输入输出

- **void SetWindowTitle(string title)**

- 设置窗口名称 (genlib.h: typedef char \*string;)

- **string GetWindowTitle(void)**

- 获得窗口名称

# 图形坐标系与像素

## ■ 图形坐标系

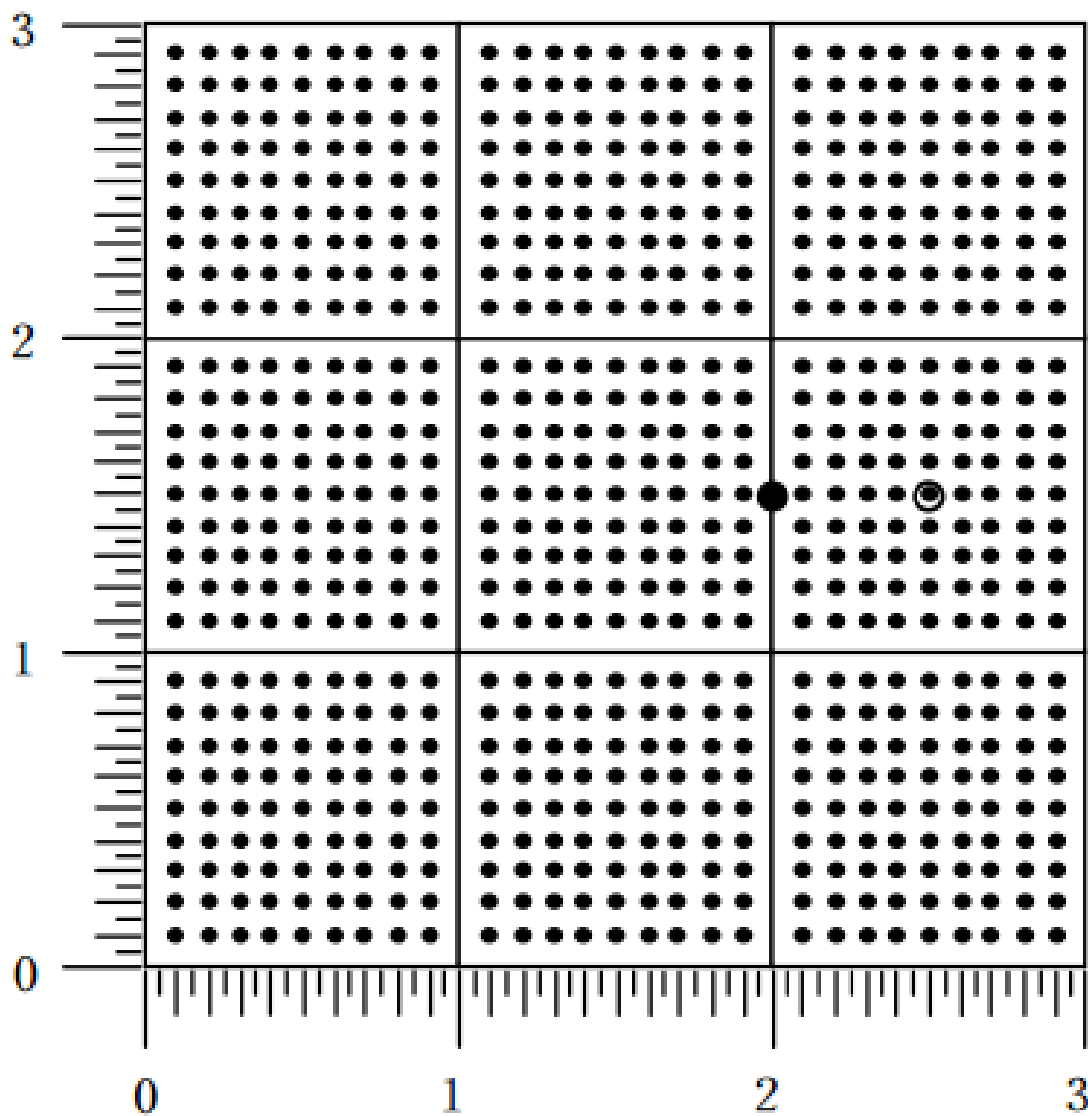
- Y轴从下往上
- 图形绘制坐标

## ■ 像素坐标系

- Y轴从上往下
- 鼠标事件位置

## ■ 相互转换

- ScaleXInches



## 窗口相关函数 - 2

- void **SetWindowSize**(double w, double h)
  - 设置窗口的大小 (InitGraphics函数调用前)
  - w - 窗口宽度, 单位英寸
  - h - 窗口高度, 单位英寸
  - 如果设置的尺寸大于屏幕尺寸, 那么系统会进行等比例的缩小, 使得符合屏幕大小
- double **GetWindowWidth**(void)
  - 获得窗口宽度
- double **GetWindowHeight**(void)
  - 获得窗口高度

# 窗口相关函数 - 3

- 获得整个屏幕的宽度和高度
  - double `GetFullScreenWidth`(void)
  - double `GetFullScreenHeight`(void)
- 获得屏幕分辨率 (1英寸像素数目)
  - double `GetXResolution`(void);
  - double `GetYResolution`(void);
- 像素与英寸转换 (图像空间 → 物理空间)
  - double `ScaleXInches`(int x);
  - double `ScaleYInches`(int y);
  - 像素坐标系 → 图形坐标系

# 基本绘图函数 - 画笔

- 想象在图形窗口里有一只虚拟的画笔存在
- 设定画笔的位置 (坐标)
  - `void MovePen(double x, double y)`
  - 坐标x和y是图形窗口的绝对坐标 (单位: 英寸)
  - `MovePen(x, y)` 将把画笔移到(x, y) – 画笔当前位置
  - 接下来的图形绘制都是从该位置开始
  - 有的绘图函数可以更改画笔当前位置
- 获得画笔的位置
  - `double GetCurrentX(void);`
  - `double GetCurrentY(void);`



# 基本绘图函数 - 直线

- void **DrawLine**(double dx, double dy)
  - dx和dy是相对于画笔当前位置的偏移量
  - 假设画笔当前位置是(x, y)，则该函数从(x, y)到(x+dx, y+dy)画一条直线
  - 画完直线后，画笔当前位置移到(x+dx, y+dy)
- 举例：画一个矩形

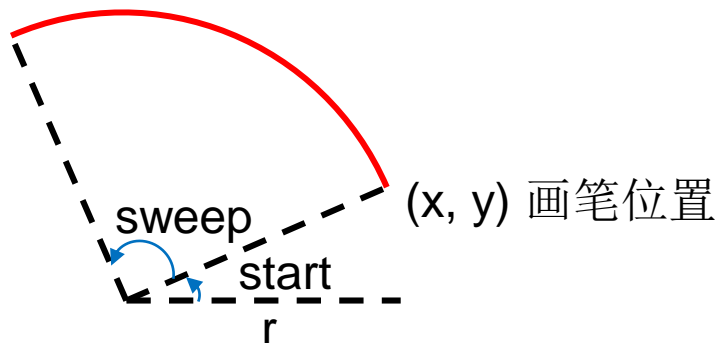
```
MovePen(0.5, 0.5);  
DrawLine(0.0, 1.0);  
DrawLine(1.0,0.0);  
DrawLine(0.0,-1.0);  
DrawLine(-1.0,0.0);
```

问题：画完矩形后，画笔在什么位置？

# 基本绘图函数 - 圆与圆弧

■ void **DrawArc**(double r,  
double start, double sweep)

- 以画笔当前位置作为圆弧所在圆的X轴上右起点，画一段圆弧
- 圆弧的半径为r，起始角度为start (单位：度，相对于X轴方向逆时针为正)，弧度为sweep

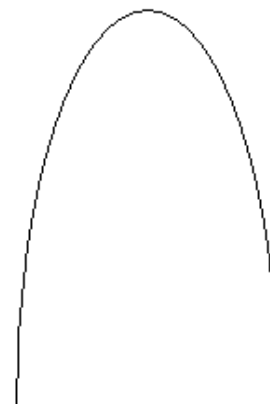


# 基本绘图函数 - 椭圆与椭圆弧

■ void **DrawEllipticalArc**(double rx, double ry, double start, double sweep)

- 以画笔当前位置作为椭圆圆弧所在椭圆的X轴上右起点，画一段椭圆圆弧
- 椭圆圆弧的两个半径分别为rx和ry，起始角度为start (单位：度，相对于X轴方向逆时针为正)，弧度为sweep

```
MovePen(width / 2, height / 2);  
DrawEllipticalArc(1, 3, 0, 180);
```



# 基本绘图函数 - 文本

- 常用的**printf**用于标准输出(控制台窗口)输出格式化数据，不能用于在图形窗口输出文本
- 图形库提供了专门用于图形窗口输出文本的函数
  - **string**是字符串指针
    - `typedef char *string; (genlib.h)`
  - **void DrawTextString(string text)**
    - 从当前位置开始输出文本(字符串)**string**
  - **double TextStringWidth(string text)**
    - 当前字体和字号下的字符串长度

# 基本绘图函数 - 文本

<http://www.cplusplus.com/reference/cstdio/sprintf/>

- **DrawTextString**函数只能输出文本(字符串), 不能直接输出格式化数据
- 函数**sprintf()**可将格式化数据输出到一个缓冲区中, 形成一个字符串
  - `sprintf(string, "format string", values...);`
- **sprintf()**的用法同**printf()**
  - `char str[20]; sprintf(str, "Hello World!\n%d", 100);`
  - **printf**将结果输出到标准输出设备(显示终端)上
  - **sprintf**将结果输出(保存)到内存缓冲区
- 从字符串输入是**sscanf()**

# 基本绘图函数 - 图形窗口控制

## ■ 绘图控制

- 清屏: void **DisplayClear**()
- 更新: void **UpdateDisplay**(void)
- 暂停: void **Pause**(double seconds)

## ■ 擦除模式 (相当于用白色绘图)

- void **SetEraseMode**(bool mode)
- bool **GetEraseMode**(void)

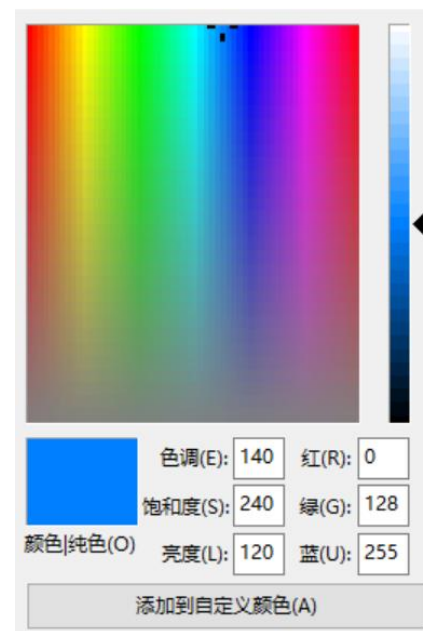
## ■ 图形绘制状态

- 结束: void **ExitGraphics**(void)
- 保存: void **SaveGraphicsState**(void)
- 还原: void **RestoreGraphicsState**(void)

# 基本绘图函数 - 属性1

■ void **DefineColor**(string name, double red, double green, double blue)

```
static void InitColors(void)
{
    nColors = 0;
    DefineColor("Black", 0, 0, 0);
    DefineColor("Dark Gray", .35, .35, .35);
    DefineColor("Gray", .6, .6, .6);
    DefineColor("Light Gray", .75, .75, .75);
    DefineColor("White", 1, 1, 1);
    DefineColor("Brown", .35, .20, .05);
    DefineColor("Red", 1, 0, 0);
    DefineColor("Orange", 1, .40, .1);
    DefineColor("Yellow", 1, 1, 0);
    DefineColor("Green", 0, 1, 0);
    DefineColor("Blue", 0, 0, 1);
    .....
}
```



DefineColor("Light Blue", 0, 128.0/255.0, 1);

graphics.c: 1835~1852

# 基本绘图函数 - 属性2

## ■ 点的大小

- void **SetPointSize**(int size)
- int **GetPointSize**(void)

## ■ 画笔的大小

- void **SetPenSize**(int size)
- int **GetPenSize**(void)

## ■ 画笔的颜色

- void **SetPenColor**(string color)
- string **GetPenColor**(void)



# 基本绘图函数 - 属性3

## ■ 区域填充

□ void **StartFilledRegion**(double density);

■ density - [0, 1] (1 - 不透明, 0.5 - 半透明, 0 - 透明)

□ void **EndFilledRegion**(void);

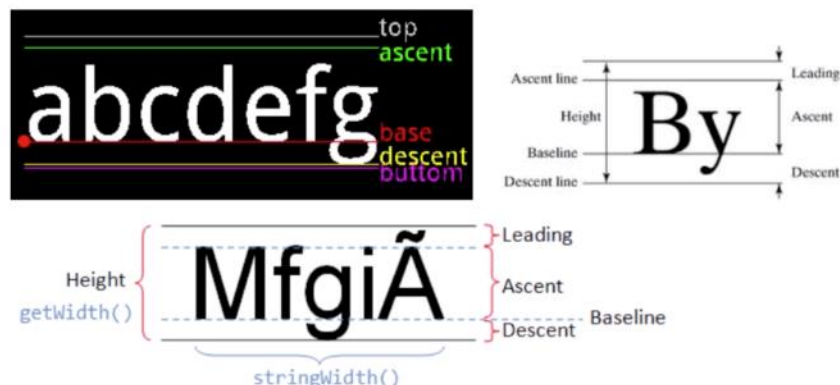
■ 举例

```
void fillRectangle(double x, double y, double w, double h)
{
    MovePen(x, y);
    StartFilledRegion(1); // 开始
    DrawLine(0, h);
    DrawLine(w, 0);
    DrawLine(0, -h);
    DrawLine(-w, 0);
    EndFilledRegion(); // 结束
}
```

# 基本绘图函数 - 属性4

## ■ 文本的字体

- void **SetFont**(string font)
- string **GetFont**(void)



## ■ 字体的样式

- void **SetStyle**(int style)
- int **GetStyle**(void)

```
#define Normal 0
#define Bold 1
#define Italic 2 exgraph.h: 148~150
```

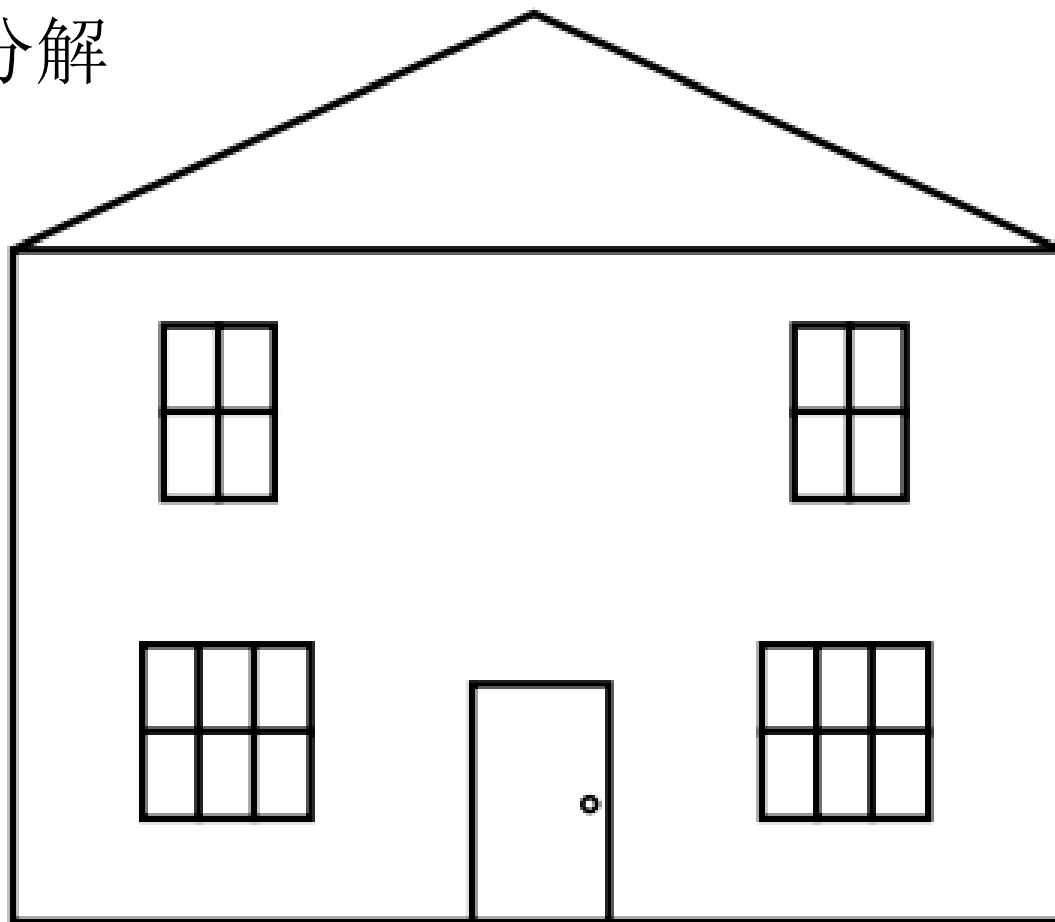
## ■ 字体的属性

- double **GetFontAscent**(void)
- double **GetFontDescent**(void)
- double **GetFontHeight**(void)

# 基本图形编程示例：画房子

## ■ 基本图形程序示例：manyones/house.c

### □ 功能分解



# 基本图形编程总结

- Win32程序 (libgraphics)
  - WinMain函数 – Main函数 – InitGraphics函数
- Win32工程创建 (Dev-C++ 和 VS2010+)
- 窗口相关函数
  - SetWindowTitle, SetWindowSize, ...
- 基本绘图函数
  - 画笔：线，(椭)圆/弧，文本，清屏， ...
  - 属性：颜色，线宽，区域填充，字体， ...
- 基本图形绘制示例
  - Main函数所在的源文件仅需#include "extgraph<sup>28</sup>.h"

# 专题三 图形程序设计

## ■ 基本图形编程

## ■ 交互图形编程

- 事件驱动编程 (事件发生 – 调用注册的回调函数)

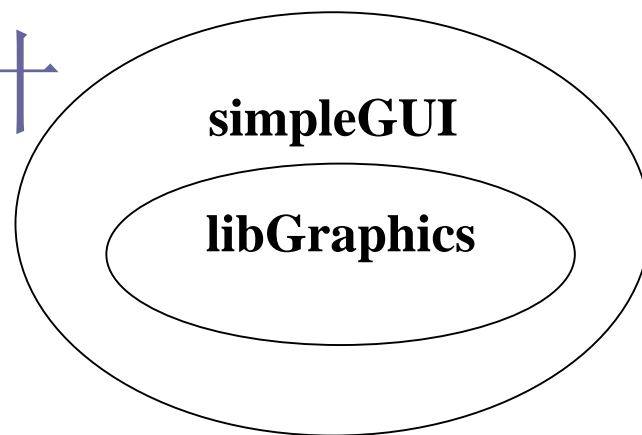
- 键盘事件 `void KeyboardEventProcess(int key, int event);`

- 字符事件 `void CharEventProcess(char c);`

- 鼠标事件 `void MouseEventProcess(int x, int y, int button, int event);`

- 定时器事件 `void TimerEventProcess(int timerID);`

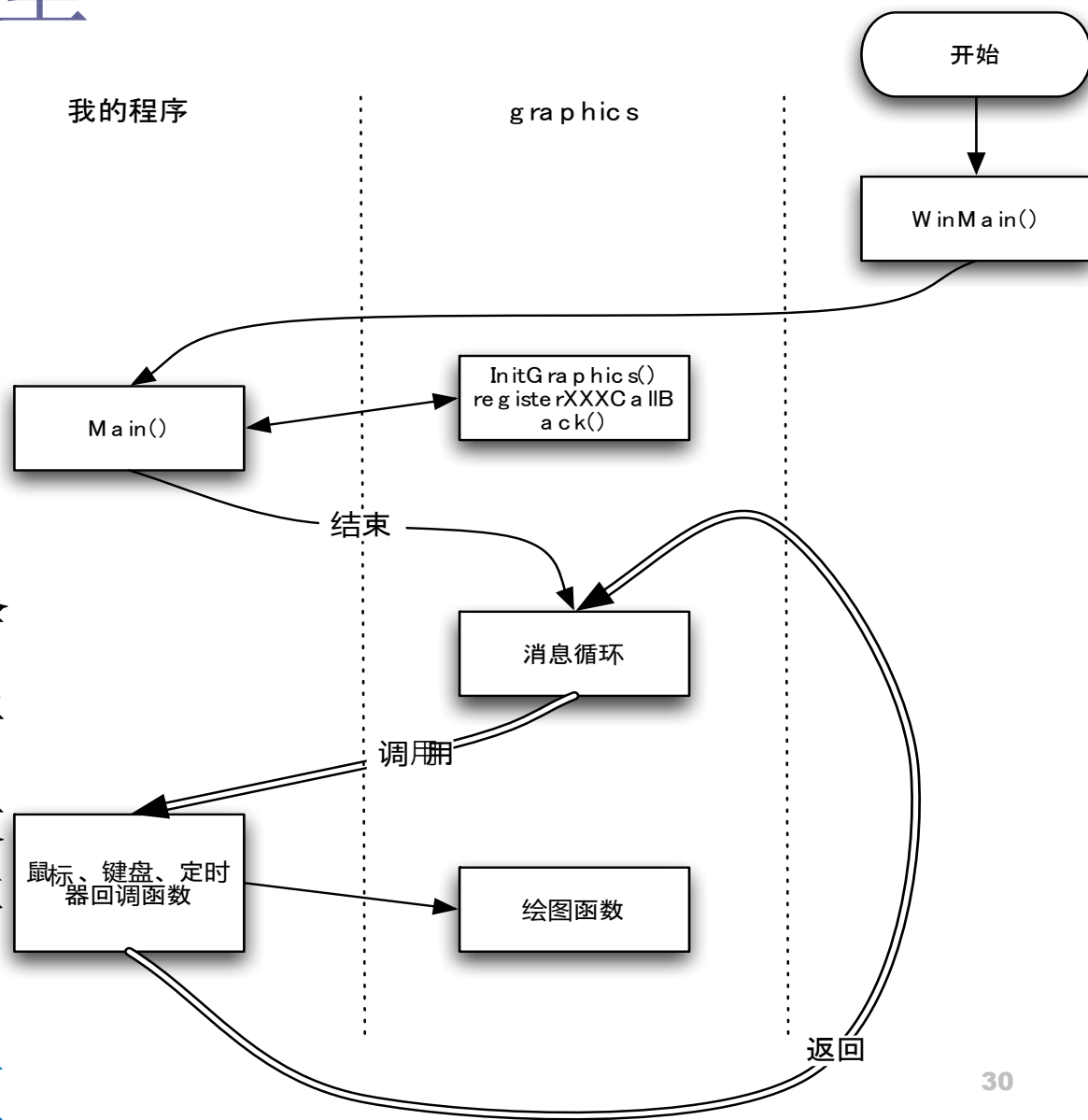
## ■ simpleGUI介绍



# 新的编程模型

- 命令行C程序  
(Console): 等待  
用户输入

- 事件驱动程序  
(Win32): 系统会  
捕获交互事件并  
把消息发给相关  
应用程序，程序  
检查消息队列，  
对消息进行响应



# 回调函数 (callback) – 函数指针

- 当事件发生时，回过来调用我的函数
    - 1. 给将来会发生事件的地方注册一个回调函数
    - 2. 当事件发生时，该回调函数被调用 (执行)
  - 回调函数经常用于事件处理 (event processing)，譬如：当按下键盘、移动鼠标等事件发生时，就调用相应的回调函数去处理这些操作
  - 可在回调函数中实现对图形的交互
    - 事件驱动编程 (1) 编写事件响应函数 (回调函数)
    - (2) 注册事件的回调函数
- 当事件发生时，Windows系统会把消息发送给程序的消息队列，程序根据消息类型，调用相应的回调函数

10:32 杭州东	G180 6时45分	17:17 北京南
商务: 候补	一等: 候补	二等: 候补
11:34 杭州东	G170 6时15分	17:49 北京南
商务: 候补	一等: 候补	二等: 候补
复兴号		
12:46 杭州东	G198 6时26分	19:12 北京南
商务: 候补	一等: 候补	二等: 候补

# 关于交互的四类回调函数原型

## ■ 键盘消息回调函数

```
void KeyboardEventProcess(int key, int event);  
/*key表示哪个键，event表示按下或松开等事件*/
```

## ■ 字符消息回调函数

```
void CharEventProcess(char c);  
/*c表示按键的ASCII码*/
```

## ■ 鼠标消息回调函数

```
void MouseEventProcess(int x, int y, int button, int event);  
/*x, y位置坐标，button哪个键，event按下/松开/移动等事件*/
```

## ■ 定时器消息回调函数

```
void TimerEventProcess(int timerID);  
/*timerID定时器号-哪个定时器触发了消息*/
```



# 回调函数类型

- 定义键盘消息回调函数指针类型

```
typedef void (*KeyboardEventCallback) (int key, int event);
```

- 定义字符消息回调函数指针类型

```
typedef void (*CharEventCallback) (int key);
```

- 定义鼠标消息回调函数指针类型

```
typedef void (*MouseEventCallback) (int x, int y, int  
button, int event);
```

- 定义定时器消息回调函数指针类型

```
typedef void (*TimerEventCallback) (int timerID);
```

# 键盘 (Keyboard)



## ■ 回调函数原型

- `typedef void (*KeyboardEventCallback) (int key, int event);`  
*/\*key表示哪个键，event表示按下或松开等事件\*/*

## ■ 注册键盘消息回调函数

- `void registerKeyboardEvent(KeyboardEventCallback callback);`  
*/\*注册键盘消息回调函数——告诉系统用哪个函数来处理键盘消息\*/*

```
typedef enum { /*键盘按键状态*/  
    KEY_DOWN,  
    KEY_UP  
} ACL_Keyboard_Event;
```

10:32 过 杭州东	G180 蓝 6时45分 ▾	17:17 绿 北京南
商务: 候补 +	一等: 候补 +	二等: 候补 +
11:34 过 杭州东	G170 蓝 6时15分 ▾	17:49 绿 北京南
商务: 候补 +	一等: 候补 +	二等: 候补 +
复兴号		
12:46 过 杭州东	G198 蓝 6时26分 ▾	19:12 绿 北京南
商务: 候补 +	一等: 候补 +	二等: 候补 +

# 键盘 (Keyboard)



**虚拟码**是一种与设备无关的键盘编码，它的值存放在键盘消息的**wParam**参数中，用以标识哪一个键被按下或释放，最常用的虚拟码已经在 **winuser.h** 中定义。 没有**VK\_0~VK\_9**和**VK\_A~VK\_Z**，直接使用字符'0'~'9', 'A'~'Z'

## 常用的虚拟码

符号常量名称	等价的键盘键或鼠标按钮	符号常量名称	等价的键盘键或鼠标按钮
VK_LBUTTON	鼠标左按钮	VK_BACK	退格键
VK_RBUTTON	鼠标右按钮	VK_TAB	制表键
VK_MBUTTON	鼠标中按钮	VK_RETURN	回车键
VK_SHIFT	Shift 键	VK_CONTROL	Ctrl 键
VK_ALT	Alt 键	VK_PAUSE	Pause 键
VK_CAPITAL	Caps Lock 键	VK_ESCAPE	Esc 键
VK_PRIOR	Page Up 键	VK_NEXT	Page Down 键
VK_END	End 键	VK_HOME	Home 键
VK_LEFT	左键头键	VK_RIGHT	右箭头键
VK_UP	上箭头键	VK_DOWN	下箭头键

键盘事件处理函数所在的源文件仅需#include <windows.h>和#include <winuser.h>

# 字符 (Char)

## ■ 回调函数原型

□ `typedef void (*CharEventCallback) (char c);`  
*/\*c表示按键的ASCII码\*/*

## ■ 注册字符消息回调函数

□ `void registerCharEvent(CharEventCallback callback);`  
*/\*注册字符消息回调函数——告诉系统用哪个函数来处理字符消息\*/*

与键盘事件的异同

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUL	32	(space)	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93	]	125	}

# 鼠标 (Mouse)



## ■ 回调函数原型

- `typedef void (*MouseEventCallback) (int x, int y, int button, int event);`  
/\*x, y位置像素坐标, button哪个键, event按下/松开/移动等事件\*/

## ■ 注册鼠标消息回调函数

- `void registerMouseEvent(MouseEventCallback callback);`  
/\*注册鼠标消息回调函数——告诉系统用哪个函数来处理鼠标消息\*/

```
typedef enum
{
    NO_BUTTON = 0,
    LEFT_BUTTON,
    MIDDLE_BUTTON,
    RIGHT_BUTTON
} ACL_Mouse_Button;
```

```
typedef enum
{
    BUTTON_DOWN,
    BUTTON_DOUBLECLICK,
    BUTTON_UP,
    ROLL_UP,
    ROLL_DOWN,
    MOUSEMOVE
} ACL_Mouse_Event;
```

# 定时器 (Timer)

## ■ 定时器回调函数类型

- `typedef void (*TimerEventCallback) (int timerID);`  
`/*timerID定时器号-哪个定时器触发了消息*/`

## ■ 注册定时器消息回调函数

- `void registerTimerEvent(TimerEventCallback callback);`  
`/*注册定时器消息回调函数——告诉系统用哪个函数来处理定时器消息*/`

## ■ 定时器启动与关闭

- `void startTimer(int timerID, int timeinterval);` `/* 开启后，循环模式 */`  
`/*启动定时器，timerID表示某个定时器，timeinterval表示定时间隔*/`
- `void cancelTimer(int timerID);`  
`/*关闭某个定时器*/`





# 相关说明

- 注册函数已在系统中定义，直接调用即可  
`registerKeyboardEvent(KeyboardEventProcess);`  
`registerCharEvent(CharEventProcess);`  
`registerMouseEvent(MouseEventProcess);`  
`registerTimerEvent(TimerEventProcess);`  
`startTimer(int timerID, int timeinterval);`  
`cancelTimer(int timerID);`

- 回调函数需要自己写

事件驱动编程 (1) 编写事件响应函数 (回调函数)

(2) 注册事件的回调函数

当事件发生时，Windows系统会把消息发送给程序的消息队列，程序根据消息类型，调用相应的回调函数

# 消息处理机制原理

## ■ WinMain函数 – Main函数

```
int WINAPI WinMain (HINSTANCE hThisInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszArgument, int nFunsterStil)
{
    MSG messages; /* Here messages to the application are saved */

    Main();

    /* Run the message loop. It will run until GetMessage() returns 0 */
    while (GetMessage (&messages, NULL, 0, 0)) {
        /* Translate virtual-key messages into character messages */
        TranslateMessage(&messages);
        /* Send message to WindowProcedure */
        DispatchMessage(&messages);
    }
    FreeConsole();
    return messages.wParam;
}
```

graphics.c: 2050~2070



# 消息处理机制原理

## ■ Main函数 - InitGraphics函数

```
void InitGraphics(void)
{
    if (!initialized) {
        initialized = TRUE;
        ProtectVariable(stateStack);
        ProtectVariable(windowTitle);
        ProtectVariable(textFont);
        InitColors();
        InitDisplay();
    }
    DisplayClear();
    InitGraphicsState();
}
```

graphics.c: 318~331

# 消息处理机制原理

## ■ InitDisplay函数

```
void InitDisplay(void)
{
    .....
    g_keyboard = NULL;
    g_mouse = NULL;
    g_timer = NULL;

    wndcls.cbClsExtra = 0;
    wndcls.cbWndExtra = 0;
    wndcls.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndcls.hCursor = LoadCursor(NULL, IDC_ARROW);
    wndcls.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wndcls.hInstance = NULL;
    wndcls.lpfnWndProc = GraphicsEventProc;
    wndcls.lpszClassName = "Graphics Window";
    wndcls.lpszMenuName = NULL;
    wndcls.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
    .....
}
```

KeyboardEventCallback **g\_keyboard** = NULL;  
MouseEventCallback **g\_mouse** = NULL;  
TimerEventCallback **g\_timer** = NULL;  
CharEventCallback **g\_char** = NULL;

graphics.c: 870~883

# 消息处理机制原理

## ■ GraphicsEventProc函数

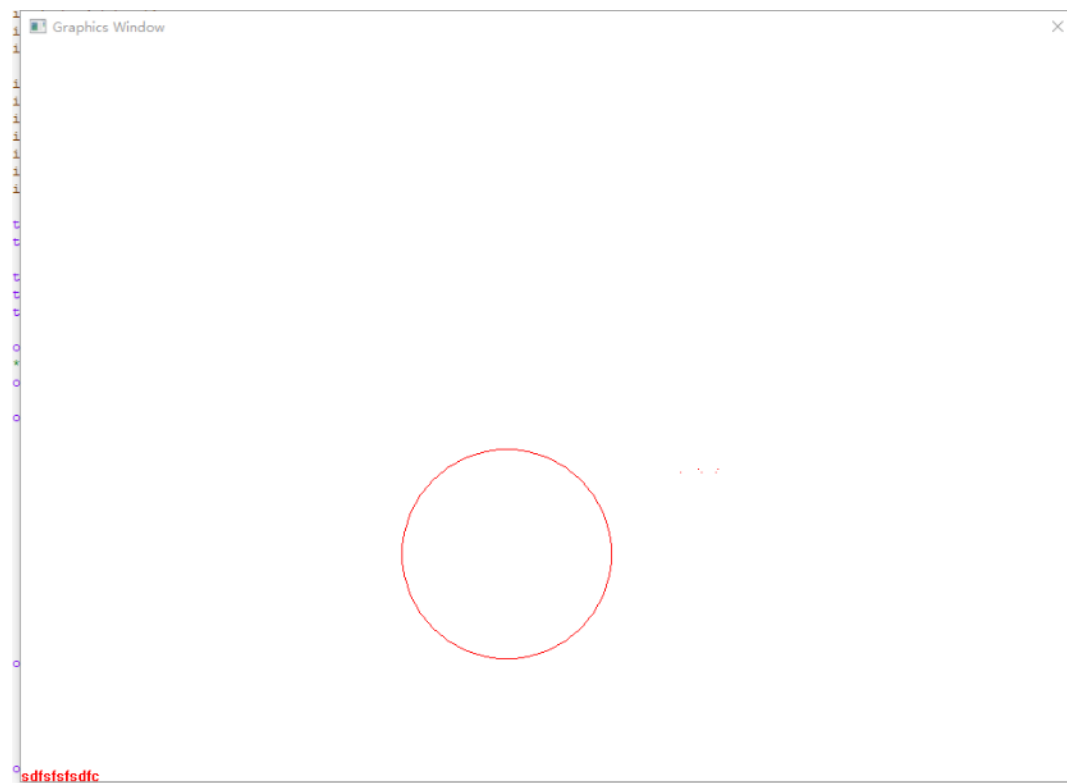
```
long GraphicsEventProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {
    switch(msg) {
        .....
        case WM_CHAR:
            if (g_char != NULL)
                g_char((char) wParam);
            return 0;
        case WM_KEYDOWN:
            if (g_keyboard != NULL)
                g_keyboard((int) wParam, KEY_DOWN);
            return 0;
        case WM_LBUTTONDOWN:
            if (g_mouse != NULL)
                g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), LEFT_BUTTON, BUTTON_DOWN);
            return 0;
        case WM_MOUSEMOVE:
            if (g_mouse != NULL)
                g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), MOUSEMOVE, MOUSEMOVE);
            .....
    }
```

```
void registerCharEvent(CharEventCallback callback) {
    g_char = callback;
}
```

鼠标移动时，未记录按键信息

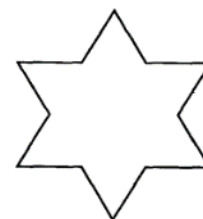
# 交互图形编程示例

- 交互图形程序示例: `manyones/igp.c`
  - 字符、键盘、鼠标、定时器交互: `igp_char.c`, `igp_keyboard.c`, `igp_mouse.c`, `igp_timer.c`

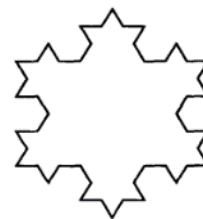


# 交互图形编程示例

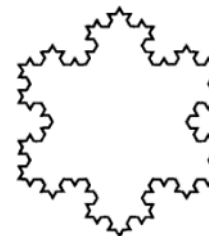
## ■ 与递归结合: `manyones/kochsnow.c`



如果你把这个图形中的每一条边再用一条有凸起三角形的线段取代, 你将得到2阶koch分形:



再次用同样的方法替换图中的每一条边, 将得到下图所示的3阶koch分形, 看起来像一片雪花:



# 交互图形编程总结

## ■ 事件驱动编程 - Win32的消息处理机制

- 键盘事件 `void KeyboardEventProcess(int key, int event);`
- 字符事件 `void CharEventProcess(char c);`
- 鼠标事件 `void MouseEventProcess(int x, int y, int button, int event);`
- 定时器事件 `void TimerEventProcess(int timerID);`

## ■ 步骤：(1) 编写事件响应函数 (回调函数)；(2) 注册事件的回调函数 (函数指针)

- 当事件发生时，Windows系统会把消息发送给程序的消息队列，程序根据消息类型，调用相应的回调函数 (阅读GraphicsEventProc函数)

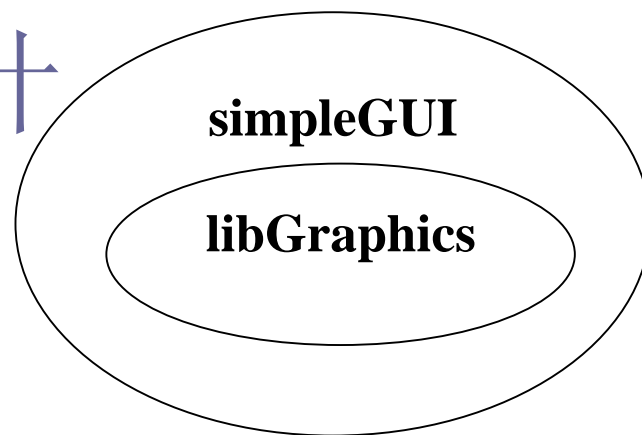
# 专题三 图形程序设计

## ■ 基本图形编程

## ■ 交互图形编程

## ■ simpleGUI介绍

- 如何使用simpleGUI - 与事件响应函数结合
- GenUIID宏 - 为GUI控件生成唯一的ID编号
- 鼠标按钮button (鼠标事件)
- 菜单列表meunList (鼠标与键盘事件)
- 编辑字符串textbox (字符事件)
- 控件颜色设置
- 其他辅助画图函数

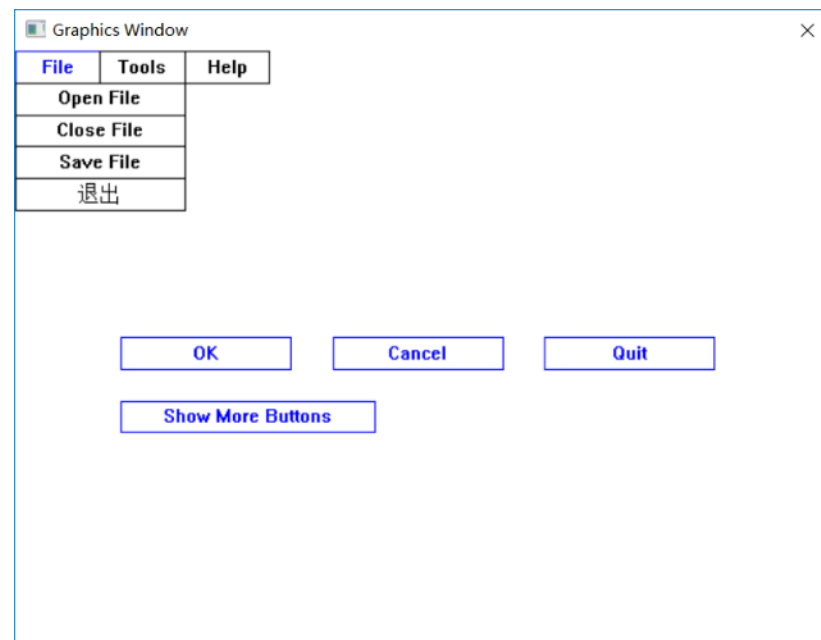


# 什么是simpleGUI

- 是一种简单的即时模式GUO
  - IMGUI – immediate mode graphics user interface
  - 适合高刷新率的应用程序
    - 屏幕总在实时刷新

- 目前只实现了三个控件

- button – 鼠标按钮
- menuList – 菜单列表
- textbox – 编辑字符串





# 如何使用simpleGUI - 1

- 必须和libgraphics库一起使用
  - 如果和其他的图形库使用，需要做简单修改
- 在程序中包含头文件 (多文件)
  - `#include "imgui.h"` (推荐)
- 将imgui.c加入程序工程中 (推荐)，或者在某个c文件中包含它 (单文件)
  - `#include "imgui.c"` (不建议)
- 所有控件的创建和响应都在display函数中完成

# 如何使用simpleGUI - 2

## ■ 首先记录鼠标和键盘输入

### □ 编写鼠标事件回调函数

#### ■ MouseEventProcess

#### ■ 调用uiGetMouse

### □ 编写键盘事件回调函数

#### ■ KeyboardEventProcess

#### ■ 调用uiGetKeyboard

### □ 编写字符事件回调函数

#### ■ CharEventProcess

#### ■ 调用uiGetChar

```
void CharEventProcess(char ch)
{
    uiGetChar(ch); /*获取字符*/
    display(); /*更新显示*/
}
```

```
void KeyboardEventProcess(int key, int event)
{
    uiGetKeyboard(key, event); /*获取键盘*/
    display(); /*更新显示*/
}
```

```
void MouseEventProcess(int x, int y,
                       int button, int event)
{
    uiGetMouse(x, y, button, event); /*获取鼠标*/
    display(); /*更新显示*/
}
```

(1) 包含头文件 `#include "imgui.h"`

(2) 鼠标、键盘和字符事件响应函数调用对应uixxx函数

(3) `display`函数创建控件，判断是否触发事件并响应事件

# 如何使用button控件

- 调用button函数创建一个按钮
- 根据返回值判断用户是否点击了该按钮，并进行相应处理

```
void display()
{
    double w = 2, h = 1, x = 0, y = 0;
    button(GenUIID(0), x, y, w, h, "OK");
    button(GenUIID(0), x += 3, y, w, h, "Cancel");
    if (button(GenUIID(0), x += 3, y, w, h, "Quit"))
        exit(-1);
}
```

```
int button(int id,
           double x,
           double y,
           double w,
           double h,
           char *label);
```

# 关于宏GenUUID - 1

■ GenUUID(N)，在编译时计算生成一个唯一号。  
。计算时使用

- 参数N
- 宏调用所在的文件名 `__FILE__`
- 宏调用所在的行号 `__LINE__`
- 宏调用时的参数N

```
#define GenUUID(N) ( ((__LINE__<<16) | (N & 0xFFFF)) ^ ((long) & __FILE__) )
```

~ 按位取反

& 按位与

^ 按位异或：相同取0，不同取1

| 按位或

# 关于宏GenUIID - 2

## ■ 用法1: GenUIID(0)

- 如果一行代码只产生一个唯一ID

## ■ 用法 2: GenUIID(k)

- 如果需要在同一行代码产生多个不同的唯一ID。例如：

```
for (k = 0; k < 3; k++)
```

```
    button(GenUIID(k), x, y-k*40, w, h, name[k]);
```

用for循环创建三个按钮，纵向排列，标签为  
names[k]

```
#define GenUIID(N) ( ((__LINE__<<16) | (N & 0xFFFF)) ^ ((long) & __FILE__) )
```

# 如何使用menuList控件 - 1

## ■ 在display函数中完成menu控件的创建和响应

- 定义菜单选项字符串 

```
char * menuListFile[ ] = { "File",  
                            "Open | Ctrl-O",  
                            "Close",  
                            "Exit | Ctrl-E" };
```

## □ 绘制和处理菜单

```
selection = menuList(GenUUID(0), x, y, w, wsub, h, menuListFile,  
                    sizeof(menuListFile) / sizeof(menuListFile[0]));  
if (selection == 3) // choose to the menu of exit  
    exit(-1);      // act on the selection
```

## ■ 用户可以用鼠标选择菜单，也可以用快捷键

- 快捷键在选项字符串中给出
- 快捷键必须是**Ctrl-X**形式，而且位于字符串的结尾部分

# 如何使用menuList控件 - 2

## ■ 控件menuList介绍

- **x, y** - 菜单左上角坐标
- **w** - 类别标签的显示宽度
- **wlist** - 菜单选项的显示宽度
- **h** - 菜单项的显示高度
- **labels** - **labels[0]**是菜单的类别名
  - **labels[1.....n-1]**是该类别菜单选项标签
  - 其中可以包含快捷键
- **n** - **labels**中标签字符串的个数

```
int menuList(int id,  
             double x,  
             double y,  
             double w,  
             double wlist,  
             double h,  
             char *labels[],  
             int n);
```

# 如何使用menuList控件 - 3

## ■ 设置动态可变菜单标签

File	Tool	Help
	Triangle	
	Circle	
	Stop Rotation   Ctrl-T	

File	Tool	Help
	Triangle	
	Circle	
	Start Rotation   Ctrl-T	

```
static int show_more_buttons = 0;  
char * menuListTool[] = { "Tool",  
                           "Triangle",  
                           "Circle",  
                           "Stop Rotation | Ctrl-T"};
```

// 设置动态可变菜单标签

```
menuListTool[3] = enable_rotation ?  
                  "Stop Rotation | Ctrl-T" :  
                  "Start Rotation | Ctrl-T";
```

```
selection = menuList(GenUUID(0), x+w, y, w,wlist,h, menuListTool,  
                    sizeof(menuListTool)/sizeof(menuListTool[0]));
```

```
if (selection == 3)  
    enable_rotation = !enable_rotation;
```

## ■ 根据实际情况设置合适的标签 (按钮与菜单)

详见demoGuiMenu.c或demoGuiALL.c



# 如何使用menuList控件 - 4


## ■ 菜单的快捷键

□ 在标签结尾设置，例如

右侧菜单列表的选项"Stop Rotation"的快捷键是**Ctrl-T** (同时按下**Control**键和字符键**t**)

那么我们将**Ctrl-T**添加到标签的末尾，注意必须在**末尾**

File	Tool	Help
	Triangle	
	Circle	
	Stop Rotation   Ctrl-T	



```
char * menuListTool[] = { "Tool", "Triangle", "Circle", "Stop Rotation | Ctrl-T" };
```


# 如何使用textbox控件，编辑字符串

- 在display函数中完成textbox控件的创建和编辑

```
static char str[80] = "Click and Edit"; // 初始化
```

```
textbox(GenUUID(0), x, y, w, h, str, sizeof(str));
```

```
int textbox(int id,  
            double x,  
            double y,  
            double w,  
            double h,  
            char textbuff[],  
            int buflen);
```

- 运行效果: 
- 如果由多个textbox，用户可以用Tab和Shift+Tab在他们之间轮转
- 根据textbox返回值判断用户是否进行了编辑
  - textbox会把用户编辑的字符串存储在形参str字符数组中

# 如何使用textbox控件，编辑字符串

- 在display函数中完成textbox控件的创建和编辑

姓名

**Text edit result is: Xing+Liu**

```
static char firstName[80] = "Xinguo";  
static char lastName[80] = "Liu";  
static char results[256] = "";
```

```
if (textbox(GenUUID(0), x, y, w, h, firstName, sizeof(firstName)))  
    sprintf(results, "%Text edit result is: %s+%s", firstName, lastName);
```

```
if (textbox(GenUUID(0), x+1.0, y, w, h, lastName, sizeof(lastName)))  
    sprintf(results, "%Text edit result is: %s+%s", firstName, lastName);
```

```
SetPenColor("Red");  
drawLabel(x, y - 1, results); // 显示结果
```

# 三类控件总结

(1) 包含头文件 `#include "imgui.h"`

(2) 鼠标、键盘和字符事件响应函数调用对应 `uixxx` 函数

(3) `display` 函数创建控件，判断是否触发事件并响应事件

- 都使用 `GenUIID` 宏生成唯一ID进行管理

- 根据控件创建函数的返回值，判断是否触发事件，并响应事件 (修改数据、重绘图形等)

- **button**

- `int button(int id, double x, double y, double w, double h, char *label);`

- 返回 `0` - 用户没有点击(按下并释放)按钮，`1` - 点击了按钮

- **menuList**

- `int menuList(int id, double x, double y, double w, double wlist, double h, char *labels[], int n);`

- 返回 `-1` - 用户没有点击(按下并释放)按钮，`>=0` - 用户选中的菜单项 index (在 `labels[]` 中)

- **textbox**

- `int textbox(int id, double x, double y, double w, double h, char textbuff[], int buflen);`

- 返回 `0` - 文本没有被编辑，`1` - 被编辑了(修改了文本，修改后的文本在 `textbuff`)

# simpleGUI控件的颜色设置 - 1

- 调用下面的函数，使用预定义的颜色组合

```
void usePredefinedColors(int k);  
void usePredefinedButtonColors(int k);  
void usePredefinedMenuColors(int k);  
void usePredefinedTextBoxColors(int k);
```

- 函数usePredefinedColors会对button/menu/textbox三种类型全部进行设置
- 而其他的三个函数对button/menu/textbox分别进行设置

# simpleGUI控件的颜色设置 - 2

```
void setButtonColors (char *frame, char*label, char *hotFrame, char *hotLabel, int fillflag);  
void setMenuColors   (char *frame, char*label, char *hotFrame, char *hotLabel, int fillflag);  
void setTextBoxColors(char *frame, char*label, char *hotFrame, char *hotLabel, int fillflag);
```

## ■ 功能

- setButtonColors - 设置按钮颜色
- setMenuColors - 设置菜单颜色
- setTextBoxColors - 设置编辑框颜色

## ■ 参数

- frame/label - 控件框/文字标签的颜色
- hotFrame/hotLabel - 鼠标划过时，控件框/文字标签的颜色
- fillflag - 是否填充背景，0 - 不填充，1 - 填充

■ 当某个参数字符串为空时，对应的颜色不做改变

■ 颜色设置是状态变量，会影响之后绘制的控件

# simpleGUI其他辅助画图函数

## ■ 画一个矩形 (x, y, w, h)

void **drawRectangle**(double x, double y, double w, double h, int fillflag);

□ **fillflag**是填充与否的标志 (1 - 填充, 0 - 不填充)

## ■ 同时画矩形和标签字符串

void **drawBox**(double x, double y,  
double w, double h, int fillflag,  
char \*label, char xalignment,  
char \*labelColor);

□ **xalignment** – 指定标签和矩形的对齐方式

■ 'L' - 靠左, 'R' - 靠右, 其他- 居中

□ **labelColor** – 指定标签的颜色名

# 图形界面程序示例

- Button: manyones/demoGuiButton.c
- Menu: manyones/demoGuiMenu.c
- TextInput: manyones/demoGuiTextInput.c
- Rotate: manyones/demoGuiRotate.c
- All: manyones/demoGuiAll.c



# simpleGUI总结

包含且仅包含头文件  
`#include <windows.h>`  
`#include <winuser.h>`  
`#include "extgraph.h"`  
`#include "imgui.h"`

## ■ 如何使用simpleGUI?

- (1) 包含头文件 `#include "imgui.h"`
- (2) 鼠标、键盘和字符事件响应函数调用对应uixxx函数
- (3) `display`函数创建控件，判断是否触发事件并响应事件

## ■ GenUIID宏 - 为GUI控件生成唯一的ID编号

## ■ 三类控件 - 根据返回值判断是否有事件发生

- 鼠标按钮`button` (鼠标事件)
- 菜单列表`meunList` (鼠标与键盘事件)
- 编辑字符串`textbox` (鼠标、键盘和字符事件)
- 控件颜色设置

## ■ 其他辅助画图函数 (`drawRectangle`, `drawBox`)

# graphics.h函数列表

```
void InitGraphics(void);  
void InitConsole(void);
```

```
void MovePen(double x, double y);
```

```
void DrawLine(double dx, double dy);  
void DrawArc(double r, double start, double sweep);
```

```
double GetWindowWidth(void);  
double GetWindowHeight(void);
```

```
double GetCurrentX(void);  
double GetCurrentY(void);
```

```
void DisplayClear();
```

// 补充的4个函数：打开/保存文件对话框，读取**bmp**图片绘制与清除

```
int OpenFileDialog(const char* filter, char filename[]);  
int SaveFileDialog(const char* filter, char filename[]);  
void DrawImage(const char* path, int xSrc, int ySrc, int wSrc, int hSrc, int xDest, int yDest, int wDest, int hDest);  
void ClearImageRegion(int xSrc, int ySrc, int wSrc, int hSrc);
```

# graphics.h函数列表

typedef enum

```
{  
    NO_BUTTON = 0,  
    LEFT_BUTTON,  
    MIDDLE_BUTTON,  
    RIGHT_BUTTON  
} ACL_Mouse_Button;
```

typedef enum

```
{  
    BUTTON_DOWN,  
    BUTTON_DOUBLECLICK,  
    BUTTON_UP,  
    ROLL_UP,  
    ROLL_DOWN,  
    MOUSEMOVE  
} ACL_Mouse_Event;
```

typedef enum

```
{  
    KEY_DOWN,  
    KEY_UP  
} ACL_Keyboard_Event;
```

typedef void (\*KeyboardEventCallback) (int key,int event);

typedef void (\*CharEventCallback) (char c);

typedef void (\*MouseEventCallback) (int x, int y, int button, int event);

typedef void (\*TimerEventCallback) (int timerID);

void registerKeyboardEvent(KeyboardEventCallback callback);

void registerCharEvent(CharEventCallback callback);

void registerMouseEvent(MouseEventCallback callback);

void registerTimerEvent(TimerEventCallback callback);

void cancelKeyboardEvent();

void cancelCharEvent();

void cancelMouseEvent();

void cancelTimerEvent();

void startTimer(int id, int timeinterval);

# extgraph.h函数列表

## // Elliptical arcs

```
void DrawEllipticalArc(double rx, double ry, double start, double sweep);
```

## // Graphical regions

```
void StartFilledRegion(double density);
```

```
void EndFilledRegion(void);
```

## // String functions

```
void DrawTextString(string text);
```

```
double TextStringWidth(string text);
```

```
void SetFont(string font);
```

```
string GetFont(void);
```

```
void SetPointSize(int size);
```

```
int GetPointSize(void);
```

```
void SetStyle(int style);
```

```
int GetStyle(void);
```

```
double GetFontAscent(void);
```

```
double GetFontDescent(void);
```

```
double GetFontHeight(void);
```

## // Style

```
#define Normal 0
```

```
#define Bold 1
```

```
#define Italic 2
```

# extgraph.h 函数列表

## // Miscellaneous functions

```
void SetEraseMode(bool mode);  
bool GetEraseMode(void);
```

```
void SetWindowTitle(string title);  
string GetWindowTitle(void);
```

```
void UpdateDisplay(void);  
void Pause(double seconds);
```

```
void ExitGraphics(void);
```

```
void SaveGraphicsState(void);  
void RestoreGraphicsState(void);
```

```
double GetFullScreenWidth(void);  
double GetFullScreenHeight(void);
```

```
void SetWindowSize(double width, double height);
```

```
double GetXResolution(void);  
double GetYResolution(void);
```

```
double ScaleXInches(int x);  
double ScaleYInches(int y);
```

## // Mouse support

```
double GetMouseX(void);  
double GetMouseY(void);
```

```
bool MouseButtonIsDown(void);
```

```
void WaitForMouseDown(void);  
void WaitForMouseUp(void);
```

## // Color support

```
bool HasColor(void);
```

```
void SetPenColor(string color);  
string GetPenColor(void);
```

```
void SetPenSize(int size);  
int GetPenSize(void);
```

```
void DefineColor(string name, double red,  
                 double green, double blue);
```

# imgui.h函数列表

```
#define GenUID(N) ( ((__LINE__<<16) | ( N & 0xFFFF))^((long)&__FILE__) )
```

```
void InitGUI();
```

```
void uiGetMouse(int x, int y, int button, int event);
```

```
void uiGetKeyboard(int key, int event);
```

```
void uiGetChar(int ch);
```

```
int button(int id, double x, double y, double w, double h, char *label);
```

```
int menuList(int id, double x, double y, double w, double wlist, double h, char *labels[], int n);
```

```
void drawMenuBar(double x, double y, double w, double h);
```

```
int textbox(int id, double x, double y, double w, double h, char textbuf[], int buflen);
```

```
void setButtonColors (char *frame, char*label, char *hotFrame, char *hotLabel, int fillflag);
```

```
void setMenuColors (char *frame, char*label, char *hotFrame, char *hotLabel, int fillflag);
```

```
void setTextBoxColors(char *frame, char*label, char *hotFrame, char *hotLabel, int fillflag);
```

```
void usePredefinedColors(int k);
```

```
void usePredefinedButtonColors(int k);
```

```
void usePredefinedMenuColors(int k);void usePredefinedTexBoxColors(int k);
```

```
void drawLabel(double x, double y, char *label);
```

```
void drawRectangle(double x, double y, double w, double h, int fillflag);
```

```
void drawBox(double x, double y, double w, double h, int fillflag, char *label, char xalignment,  
char *labelColor);
```